THE UNIVERSITY OF MICHIGAN

Technical Report 10

CONCOMP

November 1968

LOCOSS: A MULTIPROGRAMMING MONITOR FOR THE DEC PDP-7

D. R. Frantz, R. F. Brender, and J. L. Foy, Jr.

THE UNIVERSITY OF MICHIGAN

Technical Report 10

LOCOSS: A Multiprogramming Monitor for the DEC PDP-7

D.R. Frantz
R.F. Brender
J.L. Foy, Jr.

CONCOMP: Research in Conversational Use of Computers
F.H. Westervelt, Project Director
ORA Project 07449

supported by:

ADVANCED RESEARCH PROJECTS AGENCY
DEPARTMENT OF DEFENSE
WASHINGTON, D.C.

CONTRACT NO. DA-49-083 OSA-3050
ARPA ORDER NO. 716

administered through:

OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

November 1968

ABSTRACT

LOCOSS (Logic of Computers Operating System for the PDP-Seven) was developed to provide a suitable run-time environment in which to run applications programs. Multiprogramming capabilities are an essential part of the system and allow a flexible organization. Provision is made for alternate methods of establishing a separate (parallel) line of execution and for invoking a subtask. EVENTS provide a flexible means of intertask communication. A keyboard Command Interpreter provides a number of real-time control services and simple debugging aids. Input/output is buffered, overlapped, and essentially device-independent, allowing users to write programs with references to generalized "sources" and "sinks." I/O devices supported are: teletype, paper tape reader and punch, 201A dataphone, IBM 1800 (the disk file system and running programs), and the 337 display as a character sink. Additional subroutines are available for such user needs as reading and writing octal or decimal numbers, loading a program from the IBM 1800 disk, and setting an interval timer.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
LIST OF TABLES	ix
1. INTRODUCTION	1
PDP-7 System Summary	5
1800 System Summary	6
2. LOCOSS USER'S MANUAL	7
2.1 Introduction	7
2.1.1 Summary of System Components	7
2.1.2 System Communication Area	10
2.1.3 Operating Procedures	12
2.2 Multiprogramming	18
2.2.1 Basic Philosophy	18
2.2.2 Event Control	21
2.2.3 Subroutine Calls	23
2.3 Command Language Interpreter	28
2.3.1 General Commands	29
2.3.2 Disk File Control Commands	32
2.3.3 Device Assignment Commands	34
2.4 General Service Routines	38
2.5 Input/Output	45
2.5.1 Introduction	45
2.5.2 Device Descriptions	47
2.5.2.1 Paper Tape Punch	47
2.5.2.2 Paper Tape Reader	49
2.5.2.3 Teleprinter	51
2.5.2.4 Keyboard	55
2.5.2.5 IBM 1800 Logical Disk Files	61
2.5.2.6 IBM 1800 Copy Port	71
2.5.2.7 Character Generator Package	74
2.5.2.8 Dataphone	85
2.6 Buffer Management	98
APPENDIX A. LOCOSS - USER SYMBOLS	103

TABLE OF CONTENTS (cont'd)

	Page
APPENDIX B. SYSTEMS INFORMATION.....	109
TASK QUEUE.....	111
EVENT CONTROL.....	114
THE COMMAND LANGUAGE INTERPRETER.....	116
IBM 1800 COMMUNICATION ROUTINES.....	118
BIBLIOGRAPHY.....	120

LIST OF TABLES

Table		Page
1	Device Assignment Table.....	37
2	Disk Operation Codes.....	67
3	Disk Error Codes.....	69
4	Dataphone Control Characters.....	97

BLANK PAGE

1. INTRODUCTION

This is the first of three related reports describing work performed by members of the Logic of Computers Group, a research unit of the Department of Computer and Communication Sciences at The University of Michigan.

The Logic of Computers Group computer facility consists of two, small, general-purpose computers and related peripheral equipment. It is intended to provide a vehicle for heuristic investigation of problems involving large-scale simulations of generalized adaptive systems, including a large class of biologically oriented models.

This report documents those portions of the system software that are largely or completely finished, and that are not likely to undergo further substantial development. It is intended to

1. serve as a progress and research report describing the capabilities of the current software,
2. serve as a user's manual, and
3. provide enough system information to allow later users to modify or maintain the system.

While these are system types of programs, their development has been necessary to allow for further work. The LOCOSS system for the PDP-7 and the 1800 file system are basic and flexible tools. Descriptions of several other systems components are included for completeness. The particular hardware configuration is summarized at the end of this section.

In general, the TSX system provided by IBM is the basic software nucleus for the 1800. LOCOSS is the basic software nucleus on the PDP-7.

LOCOSS is the Logic of Computers Operating System for the PDP-Seven. It was developed to provide a suitable run-time environment in which to run application programs. It provides buffered, overlapped, and essentially device-independent input/output. A keyboard Command Interpreter provides a number of real-time control services and simple debugging aids. Multi-programming capabilities are an essential part of the system organization and allow flexible organization of application programs.

LOCOSS, in our estimation, provides unusually flexible capabilities and services on a machine of this size, and requires less than 2K (decimal) of core.

The availability of bulk storage on the 1800 disk via the "minor" 1800-PDP-7 interface (in use since April 1968) made it feasible to provide system programs and, perhaps more importantly, user source files, "on-line."

To implement this, a disk file system was developed for the 1800. This system provides variable-length, serial-by-character data files to both 1800 and PDP-7 users. Both symbolic and binary data are kept on-line in this manner. A keyboard utility routine on the 1800 provides simple means to load, dump, list, or copy from or to all 1800 I/O devices and the disk files. Even the approximately tripled listing rate possible with the

1050 printer (15 characters per second, hardware tabs) has been very useful. The pace of program development accelerated greatly as it became possible to be more and more disk-oriented. It was necessary to provide a new text editor because modification of the available one proved impossible. In addition to taking advantage of the device-independent I/O of LOCOSS, the editor provides a couple of string search and replacement commands that are quite useful.

The PDP-7 Assembler was adapted to accept disk file input, although it still punches object code on tape. DDT was made available on-line, and may be loaded and called from LOCOSS. A very powerful macro language, ML-I, was adapted to the disk I/O and made a part of the system. Thus program creation, editing, assembly, debugging, and execution all take place on-line under control of LOCOSS with a minimum of superfluous hard copy generation.

While the current interface is sufficiently fast for these human-oriented tasks, it will not suffice for the kind of interactive processing desired for the problem-oriented system. Therefore, the authors designed a high-speed, general-purpose interface (described in a separate report*). It offers flexibility and control substantially beyond current interfacing practice as we know it. The general ideas employed in it should

---

* Brender, R.F., and Foy, J.L. Jr., Flexible High-Speed Interface between IBM 1800 and DEC PDP-7 Computers, Technical Report 12, Concomp Project, University of Michigan, Ann Arbor, October 1968.

be very useful in other multiple computer systems (as opposed to multiple CPU systems with common memory). Implementation of thi interface should be completed by the end of 1968, and more complete reports on it will be issued later.

PDP-7 System Summary

CPU

8K of 1.75μsec core
18 bits/word
hardware interrupt

Teleprinter (33KSR)

10 char per second

Paper Tape Reader

8-channel
300 char per second

Paper Tape Punch

8-channel
63 char per second

Dataphone (201A)

synchronous
2000 bits per second
connected to switched network

CRT Display (Modified 338)

A display consisting of a DEC 338, less the PDP-8 portion of the 338, is interfaced to the PDP-7. This is locally known as a 337 and is the prototype for the DEC 339. The display operates asynchronously from instruction files in the PDP-7 memory. It provides point, increment, short vector, vector, and character plotting modes and is capable of branches and subroutining as well as conditional branches depending on the state of user-controlled switches.

1800 System Summary

CPU (1801C2)

- 16K of 2μsec core
- 16 bits/word + parity and storage protection
- priority interrupt system (12 levels)
- 3 index registers
- 1- and 2-word instruction formats
- 4 data channels

Keyboard-Printer (1816)

- 15 chars/sec

Card Read-Punch (1442)

- Read 300 cards per minute
- Punch 60 cards per minute

Disk (2310A1)

- 1 drive
- movable head
- interchangeable cartridges (2315)
- 512,000 words per cartridge

2. LOCOSS USER'S MANUAL

2.1 Introduction

LOCOSS resides in the lower part of core, locations Ø-4377 octal; all user programs are loaded above it and up to the loader area, locations 17600-17777 octal. LOCOSS is organized into four main areas: the multiprogramming monitor, the Command Language Interpreter, general service, and input/output routines. Functions within these areas are generally available as subroutine calls through a system transfer vector and communication area in low core.

2.1.1 Summary of System Components

Multiprogramming. The multiprogramming in LOCOSS is less of an attempt to provide a facility for running independent production jobs than it is an organization of system resources. On-line systems tend to be rather cumbersome if they must assume all responsibility for the user interaction while trying to execute complicated programs which are sometimes hopelessly interdependent. By providing independent lines of execution and some means of allocating resources, multiprogramming makes complex tasks easier, at least conceptually. Less complicated programs may be run as a single task under the monitor, and the user does not have to worry about the basic system organization.

The standard method of starting a user program, once it has been loaded, is to INSERT it into the task queue as an

independent line of execution from the Command Language Interpreter. From then on the program may choose its mode of operation.

Command Language Interpreter. The Command Language Interpreter (CLI) is a part of the teletype keyboard package that accepts commands in real time from the user and acts upon them. Since the CLI is essentially an interrupt-time package it cannot be "locked out" by task time routines, unless these routines fail to observe the standard system convention of operating with "interrupts on" (ION). From the CLI, the user can direct such operations as: starting up new lines of execution (tasks), dumping or changing core locations, controlling the state of the IBM 1800-interrupt service routines for the PDP-7, loading new programs from the disk, and assigning specific devices to the generalized I/O ports.

General Service Routines. The user-oriented routines provide services that are required in practically all programs as, for example, reading and writing octal or decimal numbers, outputting text lists, setting an interval timer, etc.

Input and Output. Input/output is interrupt-driven and is buffered and overlapped as much as possible. Users are expected to operate their programs with interrupts on and to perform all I/O (except special display handling) through the system routines, although there is provision for reassigning

the management for a particular device if the user so desires. Character buffering for all devices is performed by routines which are available to the user for specialized application.

The "devices" currently supported (through LOCOSS-3) are: teletype (keyboard input, teleprinter output), paper tape reader and punch, 201A dataphone (simulated full duplex), IBM 1800 logical disk file (read and write independently), IBM 1800 copy port (and through it the card reader/punch, console keyboard and printer, disk files, and running programs in the 1800), and the 337 as a sink for characters to be displayed on the screen. Among these, the support routines for the teletype, paper tape routines, and IBM 1800 disk and copy port routines are resident in LOCOSS. The dataphone and display routines are available as independent packages and may be loaded into core, in predesignated areas, by calling system subroutine LOAD, or by loading the appropriate binary tape.

Device Independence. Most I/O routines have identical calling sequences so that a large measure of device independence is possible. Users may write their programs with I/O references to generalized "sources" and "sinks," and then at run-time assign specific devices to these "ports" from the CLI. For example, a program which reads text, justifies and paginates it, and then writes it onto an output device, could be organized to read from "SOURC1" and write to "SINK3." Before the program is started the user may issue the CLI commands to assign the paper

tape reader (or the teletype, or a disk file, or the card reader through the IBM 1800 copy port, etc.), to "SOURC1" and the paper tape punch (or the teletype, a disk file, etc.) to "SINK3." Of course, if a program has a specific task to perform, the use of device-independent features will not usually be advantageous. In such a case, the program may call directly on the specific device desired. Device independence is useful mostly in general service programs and programs in which a choice of I/O ports is most conveniently made at run time.

Unfortunately, the attempted device independence is not complete. Disk files operating as sources or sinks must first be opened-in or -out before information may be read from or written into them. Thus, at the time a user assigns the disk to a port, he should also open the appropriate file through the CLI commands for this purpose. After a program has finished using a file it should be closed, again by the appropriate CLI command. Various disk errors may also occur (cf., Section 2.5.2.5), so that a user who anticipates that a disk file may be assigned to a port should provide for this possibility by setting up an appropriate disk error-handling routine.

2.1.2 System Communication Area

The system communication area is a set of locations in low core through which the user accesses LOCOSS subroutines and variables. Most of the system subroutines have the address of their entry points in this area. For example, to punch an alphabetic character on tape, a program calls the appropriate

subroutines indirectly through location 37 (octal), thusly: "JMS I 37". This "absolute transfer vector" type of access to system subroutines insulates user programs from the vagaries of system changes, so that everyone need not change his program when a minor system change is made.

A symbolic tape (and disk file) define all the communication area locations as instruction-like mnemonics. Using the definitions from this tape, then, in the previous example the user may write only "PCHA" since the tape makes the definition "PCHA JMS I 37". Thus by making the LOCOSS symbols the first tape in an assembly the user may have symbolic access to the communication area. The tape also contains the symbolic operation codes for the IBM 1800-PDP-7 interface. (See the disk and copy port sections for use.) Note that this tape should come before any user tapes in an assembly since it "FIX"es the symbol table so that none of the LOCOSS definitions appear in the user's symbol table printout. A listing of this tape appears in Appendix A.

In addition to indirect jump and subroutine jump entries in the communications area, there are also directly referenced system variables. Some of these variables may be set by the user to control the action of some subroutines. For example, the variable LEDING (location 130 octal) contains the ASCII character which is used to replace leading zeros during the output of a decimal or octal number. To indicate that the system routines should output a number with leading

blanks, the user puts an ASCII blank into the indicator location:

```
LAC     (240)
DAC     LEDING
```

The generalized sources and sinks are also addressed in this fashion and may be set directly by a user program. The CLI instructions for assigning specific devices to the ports puts an effective JMS into these locations, so that the normal method of using these ports is via an execute instruction; for example, XCT SINK2. There are three each of the generalized sources and sinks.

Other directly addressed locations contain pointers to information within LOCOSS which may be of possible interest to the user. For example, RDRLOC contains the location of the system buffer control block (BCB) for the paper tape reader. If a user wishes to provide a larger reader buffer than is provided by the system, or wishes to play games with the information after it enters the buffer, he may choose to change the BCB through RDRLOC.

2.1.3 Operating Procedures

Start-Up. Two procedures are available for loading LOCOSS. If the IBM 1800 is not prepared to service PDP-7 requests, the system must be loaded entirely from the tape labeled LOCOSS-3 (found in the trays of system tapes). Place it in the reader, set the ADDRESS switches to 17720 octal and press READ-IN. The

tape will proceed to load itself and start automatically. A halt at location 17756 indicates a checksum error on reading the tape. An adventuresome user may press CONTINUE to ignore the error, but the recommended procedure is to start over. The above sequence will be referred to as "CORE IMAGE LOADING", or the CIL procedure.

If the IBM 1800 is prepared to service the PDP-7, the following, faster, sequence may be followed. Place the tape labeled "LOCOSS BOOTSTRAP" in the reader, set the ADDRESS switches to 17777 octal and press READ-IN. This short tape loads LOCOSS from the 2310 disk and starts it automatically. A halt at location 17642 indicates a checksum error, and may be ignored by a CONTINUE. A halt at location 17772 indicates a disk error in the loading procedure, in which case the error code is in the accumulator.

Loader Area. Both of these methods of loading use non-DEC-standard loaders and preempt the normal loader area. If the user wishes to use the RIM or FF loaders, he must load them via the normal tape procedures or use the LOCOSS command "RIM" to load them from the disk. When LOCOSS (or, in fact, any program) is loaded from the disk, the loader used occupies locations 17600-17777. Unless this copy of the loader is destroyed it may be reused directly by any program. The normal method for loading new programs from the disk is via subroutine LOAD. If, however, a user program is not LOCOSS-compatible (and perhaps destroys LOCOSS), it may use the loader in high

core. It should also be noted that copies of the two routines DSKERR and DISK are included in the loader and may be called by the user for non-standard uses (e.g., when the LOCOSS routines are not in core). Details on these procedures may be obtained from the loader write-up.*

As stated above, the loading procedures for LOCOSS produce an automatic start. The system may be completely restarted at location 22 (octal) either manually or by transferring control from a program. Location 22 contains a CAF (Clear All Flags) instruction so that all I/O will be aborted, the display stopped, and the dataphone hung up. Transferring control from a program to location 23 will restart the system without affecting the display or dataphone run status.

Core Storage. The resident section of LOCOSS occupies locations 0-4377 octal, and the loaders occupy locations 17600-17777 so that the user area is 4400-17577. If the user wishes to use the character generator or dataphone routines he must load them independently into high core.

Since the display requires storage in the lower half of core for a push-down stack, locations 4370-4377 of LOCOSS are reserved for this purpose, providing a four-level subroutine capability before overflowing into the normal user area at

---

* Brender, R.F., Frantz, D.R., Foy, J.L. Jr., and Schunior, T.W., Specialized System Software for Interacting DEC PDP-7 and IBM 1800, Technical Report 11, Concomp Project, University of Michigan, Ann Arbor, October 1968.

4400. Of course, the user may also choose to establish the push-down area anywhere else in his section of lower core.

Locations 3 and 6 are available to all programs as volatile, temporary storage locations. Both locations are saved and restored by the interrupt service program so that they are available to interrupt-time as well as task-time routines. In general, any call to a LOCOSS subroutine will result in the destruction of these locations. Also, any call to a LOCOSS subroutine will result in the destruction of the active registers (AC, MQ, and L) except as noted in the specific calling sequences. Index registers 16 and 17 are reserved for system use, but registers 10-15 are available to the user.

The user has access to the reader and the punch buffer control blocks in LOCOSS through locations RDRLOC and PCHLOC in the communications area. He may change these blocks to point to a larger area in user core if he so desires, but before completion, the program should restore these blocks to their former state, either directly or by reloading LOCOSS.

In summary, while operating under LOCOSS the locations that the user's program may reference are: 3,5,10-15, the communications area (about 24 to 130), the reader and punch buffer control blocks, 4370-17577, and selected portions of the loaders.

Run-Time Conventions. Since most I/O is performed under interrupt control, normal operation is with interrupts on, although interrupts may be turned off for short periods to perform delicate operations. The interrupts normally handled by the resident system are keyboard, teleprinter, reader, punch, and clock. When the dataphone routines are in core and initialized, the system will handle dataphone transmit and receive interrupts. When the display character routines are in core, they will handle the external stop and manual interrupts. All other interrupts will cause a halt. Thus the user should ensure that the handler for each interrupt that may occur is in core and initialized through subroutine INTABL.

Users should acquaint themselves with the philosophy of multiprogramming and then consider the use of BUSY exit in the middle of very long tasks or in the case of blocking conditions. A BUSY enables the appearance of parallel processing, allows other tasks to gain control for a time, and avoids "lock-outs" of CPU control. The normal method of ending a program is via a "DELETE" exit, rather than a halt. This allows all buffered I/O to terminate normally.

Error Messages. Error messages report on intolerable conditions within the executive. When one of these conditions is detected, all task processing is halted, a CAF instruction is given, the error number is printed, preceded by the letters "@ERR," and LOCOSS is completely reinitialized.

If the same error is encountered twice in succession, the computer will halt with the error number in the AC. (This feature protects against looping before an error message may be printed.) Pressing CONTINUE in this case produces the same effect as starting at location 22. The following is a list of the error numbers.

000001	There was no room in the task queue when some program called INSERT, SUBTSK, BUSY, or WAIT.
000002	There was no room in the event table when some program called CREATE.
000003	A disk error occurred on a call to DISK, DSKGET, or DSKPUT when no previous call or DSKERR had been made. Location 20 (octal) contains the error number.
000004	There was no room in the clock table when some program called CLOCK.
2XXXXX	A CAL trap occurred from location XXXXX.

Operating parameters. In the current system there is room for ten decimal tasks, eight events, and four simulated user clocks. The reader and punch buffers are both of size ten and may be "packed." The teleprinter buffer has size fifteen and may be packed. The keyboard buffer will accept seventy-four characters.

2.2 Multiprogramming

2.2.1 Basic Philosophy

Multiprogramming is an organization of system resources which aids in the conceptualization of some programs by enabling the user to start more than one job or line of execution at a time. The LOCOSS multiprogramming system also includes features to facilitate intra-program communication and resource allocation. All of LOCOSS I/O uses the multiprogramming system to great advantage, although the user may run programs under LOCOSS in almost complete ignorance of the system organization. The following is a summary of system philosophy and a tutorial in system use.

In general, a section of code, a subroutine, will be called a program. It usually operates on some parameters to produce a desired result. An invocation, or instance, of a program, with parameters, is called a task. In a multiprogramming environment, several tasks may be operating in parallel; i.e., if there were more than one processor, these tasks would be executing simultaneously. With one processor these tasks are executed serially, but at certain points control is switched between one task and another so that each gets a portion of the serial time, and the total state of the system after many such task-switchings approximates the parallel processor case. In timesharing systems, this switching takes place after a given amount of time. In LOCOSS, the switching takes place only when (but not always when) control is returned to the executive

(e.g., I/O calls, event control calls, etc.).

In general, once a section of code starts execution as a task, it will change itself and the data it is working on so that it must proceed to its conclusion completely before being able to be restarted as another task. Since several tasks may wish to call the same program at once, that program must have some way of indicating its readiness to start functioning as a task. Taking advantage of the subroutine calling structure of the PDP-7 we will let the entry point of a program signal the attribute of enterability. If the program is ready to begin executing as a task, its entry point will be zero. If it is not able to start executing as a new task, its entry point will be nonzero. Since a call to a subroutine automatically makes the entry point non-zero, all that is required for a program at the end of one execution of a task to indicate its readiness to begin another task is to zero its entry point before returning. For example, a task which returns to its caller might appear as:

```
SUB, Ø
     ____              /first executable instruction
      .
      .
     LAC SUB
     DAC TEMP          /save return pointer
     DZM SUB           /ready to restart
     JMP  I  TEMP      /return to caller
```

A task may start up other tasks in two ways. The first is by starting a parallel line of execution by calling system subroutine INSERT. After having been started, the second

task will not in general depend on or be related to the first, or vice-versa. They may be completely independent. As an option for INSERTing tasks which are known always to be re-enterable, the call to INSERT may indicate that control is to be transferred directly to the indicated entry point rather than transferring control as a subroutine call; i.e., entry will be via JMP rather than JMS, and enterability will not be checked.

The second method of starting a task is to call on it as a subtask by calling system subroutine SUBTSK. A program called as a subtask is expected to accomplish some action and then return to its caller. The calling task remains dormant until its subtask finishes, and then regains control upon return from the subtask. A subtask in turn is a task and may start other tasks, and so forth.

When a request is given to the system to start a task, there are usually other similar outstanding requests. These requests are kept in the TASK QUEUE until such time as these tasks may be started. A task is added to the end of the queue so that all other outstanding tasks are started first (if they are enterable). This feature allows the semblance of parallel processing. Eventually a request comes to the top of the queue for consideration. If a program is not enterable, the request goes to the end of the queue to be tried again later. When a program is enterable, the information regarding its call is removed from the queue and the call is made as if it came directly from the calling task, or, in the case of an INSERTed task, as if it were starting anew.

When a task teaches a temporary impasse, such as a wait for input or a similar occurrence over which it has little control, it should take a BUSY exit, i.e., it should return to the executive and give some other task a chance to execute while the blocking condition clears up. The executive will put information regarding the busy task at the end of the queue and return to that task after the others in the queue have had their chance to execute. The task can then check the condition and either go on or return itself to the queue again by another BUSY exit.

When an independent line of execution (i.e., one which is not a subtask, but was started by an INSERT) finishes, it need not return to a calling task but may rather return control to the executive by a DELETE exit. A task which was INSERTed as a JMS entry may also return in the normal way. Both methods of return have the same effect—removing the task from the active roll. The task should of course zero its entry point before the exit. Note that if a program returns as if it were called via JMS, first zeroing its entry point (as in the example above), it may be called as a subroutine or a subtask, or inserted as an independent task.

2.2.2 Event Control

In a multiprogramming system with independent lines of execution there sometimes exists the problem of coordinating parallel tasks. Two otherwise essentially independent tasks

may need to signal each other that certain points in operation have been reached, since one may have to wait until the other has finished some processing. In particular, it would be difficult to ensure that the passing of parameters to an independent, INSERTed task had taken place properly if there were no task-task means of communication.

Events are offered to resolve this problem. Events are extra-task entities which a task may CREATE for some purpose as above. Each created event has a unique name (a number) which may be given to other tasks for reference. The task may then take a WAIT exit, that is, return itself to the queue until the given event has occurred. This task in the queue is not reactivated until the event has occurred. One of the daughter tasks, in the meantime, has the responsibility of giving the SIGNAL that the event has been completed. After this, the next time the waiting task comes up for consideration in the queue, it will be reactivated and can continue safe in the assurance that the blessed event is properly cared for.

In view of the foregoing discussion, we may make the following general comments about system operation. Most user programs will be written as at least one task. Thus, a standard method of starting a program when LOCOSS is in core is: first load the user program tape and start execution at location 22 (either by a START 22 on the program tape or by a PAUSE on the tape followed by manually pressing the START key with the ADDRESS switches at 22). Either way, starting at 22

initializes LOCOSS. Then type "(ESC) INS XXXXX" which is the CLI command to insert a task into the queue at location XXXXX, the entry point of the user's task. Recall that this entry point should be zero for the task to be started. If entry is to be made via JMP, the proper command is "(ESC)INS 4XXXXX".

If the user task does any I/O, it should be recalled that, with high probability, some output buffer will eventually fill or some input buffer will be empty. If either of these conditions occurs, the I/O task segment will take a BUSY exit. That is to say, every other task in the queue will be given a chance to execute before control is returned to the BUSYed task. If there is only one task in existence, there is no problem. However, when there is more than one line of control, care should be taken that any I/O referencing is coordinated among them so that two tasks will not be calling an I/O subroutine at the same time with an undesired interleaving of the transferred information and loss of correct return points from the I/O program.

2.2.3 Subroutine Calls

SUBTSK

Purpose: To put a task into the queue and to relinquish control of the current task until the subtask is finished.

Calling Sequence: AC = argument

SUBTSK

Calling Sequence (cont'd):

```
                ROUTIN
                .
                .       /optional arguments
                .

        RETURN, -------
```

ROUTIN is the address of the task to started

RETURN is the location to which control is transferred when the task at ROUTIN is finished.

Optional arguments are to be consistent with the called task's expectations. When ROUTIN is given control the AC is the same as upon the call, although the MQ register may have been destroyed. The entry point ROUTIN itself contains the address of the word after ROUTIN in the calling sequence above, as if the call were a direct JMS from the location containing ROUTIN, so that the return sequence in the first example in Multiprogramming (above) may be used. Upon entry to SUBTSK interrupts may be on or off.

INSERT

Purpose: To insert an independent task into the queue.

Calling Sequence:

```
                IOF
                AC = argument
                INSERT
                ROUTIN      / or LAW ROUTIN or 400000+
                              ROUTIN (See option below.)
        RETURN, ION
```

ROUTIN is the entry point of the task to be started. If "LAW ROUTIN" is used in the call, entry is made via JMP rather than JMS. RETURN is the point to which control is returned immediately after the task is entered into the queue. When task ROUTIN is given control, the AC is the same as upon the call although the MQ may have been destroyed. In a JMS call, location ROUTIN itself is set nonzero to indicate that the task is not enterable. Such independent tasks may relinquish control by DELETE or by returning as in SUBTSK. Both methods return control to the same location in the Multiprogramming monitor. A task entered via JMS should zero its entry point before either method of return.

BUSY

Purpose: To return the currently executing task to the queue to be reactivated later.

Calling Sequence: AC = any

```
        BUSY
RETURN, ----
```

Control is returned to location RETURN when the task next comes under consideration in the queue. The AC is saved and restored around the call to BUSY by the system. Interrupts may be on or off upon entry. Return is made with interrupts on.

DELETE

Purpose: To return control to executive when an independent line of execution is finished.

Calling Sequence: DELETE

Interrupts may be on or off upon entry.

CREATE

Purpose: To obtain a unique event number for reference use.

Calling Sequence:
```
IOF
CREATE
RETURN, ———
```

When control is returned to location RETURN the AC contains a unique event number for reference in WAIT or SIGNAL calls.

WAIT

Purpose: To return the currently executing task to the queue until a given event occurs.

Calling Sequence:
```
AC = event number      /ION or IOF
WAIT
RETURN, ———            /with ION
```

After the event corresponding to the reference number in the AC occurs, a transfer to location RETURN is scheduled in the task queue. If the event has already occurred or if there is no event corresponding to the reference number, return is immediate.

SIGNAL

Purpose: To indicate that a given event has occurred.

Calling Sequence: AC = event number /IOF

SIGNAL

RETURN, ——— /ION

The event corresponding to the reference number is marked as having occurred and all tasks waiting for the event will be reactivated as their turn comes in the queue. No action occurs if the event has already taken place or if the event referenced is nonexistent.

CLEARE

Purpose: To reinitialize all events and event handling

Calling Sequence: IOF

CLEARE

RETURN, ———

The next event to be created will have number one (1).

CLEARQ

Purpose: To clear the task queue of all outstanding tasks.

Calling Sequence: IOF

CLEARQ

RETURN, ———

After execution of CLEARQ the only line of execution (the active task) will be the one which called CLEARQ.

2.3 Command Language Interpreter

The Command Language Interpreter (CLI) is a portion of the LOCOSS keyboard routines which provides the user with a measure of control over the immediate, real-time state of the system. Since the CLI is an interrupt-time package, a command line may be entered at any time, independently of the state of the user program.

A command line is a keyboard input line which has as its first character an ESC (Control-Shift K). Thus the line is subject to the same editing as other keyboard input. Once an ESC is typed at the beginning of a line it may not be deleted by LEFTARROW; thus, the line will be dispatched to the CLI unless RUBOUT is typed. When the carriage return is entered, finishing the command, all task processing is stopped until the command has been carried out successfully, at which time the CLI types "DONE." If any key is struck before this time the command is aborted. If a command is ill-formed, the CLI will type back "WHAT?" and ignore it. When echoing is on (see Section 2.5.2.4) an "@" is echoed when ESC is typed.

There are three groups of commands: general control, disk file control, and device assignment. The format for device assignment commands is given separately below. For the first two groups, a command is a three-letter mnemonic, possibly followed by numeric parameters separated by non-digit character strings (e.g., space), optionally followed by any character string (for comment recording purposes).

Numbers given to the CLI are octal or decimal depending on the command type. Only the eighteen low-order bits of any number are used. After the first three characters of an input line, all nondigit characters are ignored except as terminators of numbers.

2.3.1 General Commands

ATN

ATN sends an attention character over the dataphone (if it can), by calling subroutine DFNATN. The CLI message "DONE" printed at the command termination means merely that the transmission has been scheduled, not necessarily that it has been started or completed.

DDT

DDT causes control to be transferred to location 16000 (octal). Control may be returned to the program and LOCOSS by transferring control from DDT to location 2, i.e., by typing "two prime" (2'). DDT should be told to watch the teleprinter flag by giving it the TTY$ and STIOC$ command and by setting the variable TTYF$ to 1.

DAT

DAT initializes the hardware and software for dataphone communications; that is, it sets "terminal ready" on the data set and initializes the dataphone routines by calling DFNI. This command must be given before incoming or outgoing calls are made on the data set and may be repeated thereafter to clear up any bad messes without hanging up the set.

DMP

DMP takes one or two octal arguments. If there are two arguments, all the core locations between and including the two locations given by the arguments are printed on the teletype, in octal, eight words per line, with appropriate labeling. If only one argument is given, only that one location is dumped. For example, DMP 10 17 will dump all locations between and including 10 and 17 (octal).

ECF

ECF causes echoing to be turned off.

ECN

ECN causes echoing to be turned on. (See the discussion under the keyboard routines.) The original state of the system is with echoing on.

HEL

HEL merely types back this version's assembly data to assure an anxious user that the CLI, at least, is still listening to him.

HSR

HSR starts the high-speed tape reader after it has been stopped by an XOFF or end-of-tape condition, or if it has never been started. This command should not be given if none of these conditions exist. (See the high-speed reader routires.)

INS

INS takes one or two octal arguments. The first is the address of a task to be entered into the task queue, and the second, if any, is the argument to be passed in the AC to the task when it is scheduled. If there is no second argument, a zero is passed to the task, e.g., INS 4400 15.

RES

RES resumes task processing at the point at which it was interrupted by STP. If no STP was given since the last RES, a second RES has no effect.

RST

RST completely restores the pristine state of the system (probably). (See system routine RESET.)

STO

STO takes two or more octal arguments. The first is considered as a location in which to store the second argument. Additional arguments are stored in successive locations. For example, STO 4500 1 2 3 has the effect of putting a 1 into location 4500, a 2 into 4501, and a 3 into 4502.

STP

STP stops all task processing until the RES command is given. The CLI is still available during task suspension (most importantly for DMP and STO). A second STP before an RES is treated as an illegal command.

TAB

TAB takes one decimal argument and uses it to set the system variable TABCNT; that is, it sets the tab spacing. E.G., TAB 15 will set tab stops at decimal 16, 31, 46,...

2.3.2 Disk File Control Commands

The following commands provide direct communication with, and control of, the disk file servicing routine in the IBM 1800 through the system subroutine DISK.*

With the exception of LOD and RIM, they all print the return from the disk routines indicating success or failure, and in case of failure even LOD and RIM print the failure number. A success return is indicated by printing ØØØxxx (octal) where xxx is the returned value. A failure return is indicated by ØØ6xxx where xxx is the disk error code.

A CLI call on DISK should not be issued if a program is in the midst of conversing with the disk, since no effort has been made to make the routines reentrant. A CLI call will not disturb the user disk error return.

CLI

Close the currently opened input file.

---

* Brender, R.F., Frantz, D.R., Foy, J.L. Jr., and Schunior, T.W., Specialized System Software for Interacting DEC PDP-7 and IBM 1800, Technical Report 11, Concomp Project, University of Michigan, Ann Arbor, October 1968.

CLO

Close the currently opened output file.

CLR

Close any opened input or output files.

CRE

CRE takes one decimal argument and creates a file with that number.

DES

DES takes one decimal argument and destroys the file with that number. No verification of the number is sought so this command should be used carefully.

INT

INT takes one or two octal arguments and uses them as parameters for a direct call on the system subroutine DISK. The first argument is the disk operation code and the second is a file number or character. The second argument need not be present if not applicable (e.g., in a CLOSE call) and is defaulted to zero.

LOD

LOD takes one decimal argument and uses it as the parameter for a call on system subroutine LOAD; i.e., the addressed file is loaded into core. If a disk error occurs during loading, the disk error number is printed out and the loading is aborted.

OIT

OIT takes one decimal argument and opens that file for input, translated.

OIU

OIU takes one decimal argument and opens that file for input, untranslated.

OOT

OOT takes one decimal argument and opens that file for output, translated.

OOU

OOU takes one decimal argument and opens that file for output, untranslated.

RIM

RIM loads the READ-IN-MODE and FUNNY FORMAT loaders into their proper positions in core (17600-17777) from File 5. This file is in bootstrap format and cannot be loaded by a LOD command.

2.3.3 Device Assignment Commands

The device assignment commands provide the link between a program's references to generalized SOURCEs and SINKs and the actual devices used in a particular "run" of a program (see Section 2.1). For example, the assembler is written to accept input from SOURC1, and the user directs the CLI to

assign either the teletype, the tape reader, the IBM copy port, or a disk file to SOURC1 before the assembler starts.

The form of the device assignment command is:

<DEVICE ASSIGNMENT COMMAND>:: = <DEVICE> <PORT>|<DEVICE> <MODE> <PORT>

<DEVICE>:: = TTY|TAP|DSK|ICY|DFN|DSP|DMY

<MODE>:: = BIN|ALP

<PORT>:: = SC1|SC2|SC3|SK1|SK2|SK3

If no mode is given, ALP (alphanumeric) is the default case. Thus, the user may assign the appropriate input or output routine for the teletypewriter, paper tape, disk, IBM copy port, dataphone, display, or dummy routine, in binary or alphanumeric mode, to one of the system transfer vector locations SOURC1, SOURC2, SOURC3, SINK1, SINK2, or SINK3. Any illegal assignments or improper syntax will be greeted with "WHAT?" and the command will be ignored.

The following table gives the legal assignments for devices. Blank entries indicate illegal assignments, i.e., nonsensical or not-yet-coded system functions. The other entries are the LOCOSS-defined assembly instructions which are inserted into the source or sink location. For example, issuing the command "(ESC)TTY SK1" is equivalent to executing the code:

```
LAC     (TPRA)
DAC     SINK1
```

A user's program may determine whether a specified device has been assigned to a particular port by executing an SAD instruction, thusly:

```
LAC     SOURC1      /get SOURC1
SAD     (RDRA)      /compare to RDRA
——                  /SOURC1 = RDRA
——                  /SOURC1 ≠ RDRA
```

Table 1

Device Assignment Table

DEVICE	ALPHANUMERIC PORTS Source	Sink	BINARY PORTS Source	Sink
TTY	TPRA	KBDA		
TAP	RDRA	PCHA	RDRB	PCHB
DSK	DSKGET	DSKPUT		
ISY	ICYRDT	ICYWRT		
DFN	DFNRA	DFNSA		
DSP		JMS 17006		
DMY	DUMMY	DUMMY	DUMMY	DUMMY

2.4 General Service Routines

RESET

Purpose: To reinitialize the whole system.

Calling Sequence:

```
RESET      /ION or IOF
RETURN,    ———      /ION
```

All I/O is aborted, and device handlers for the keyboard, teleprinter, reader, punch, clock, disk IBM 1800 copy port, and dataphone are initialized by the system routines. The task queue is emptied and event handling is reinitialized. The system variables are reset to provide echoing and prompting characters for keyboard input, i.e., ECHOSW = PRMPTC = 0. The tab count is set to ten (decimal) and LEDING is set to an ASCII zero (octal 260). Location 1 is reset so that LOCOSS provides all interrupt dispatching.

CLOCK

Purpose: To enable a timed return to a section of the user's program.

Calling Sequence:

```
          IOF
          AC = -N       /twos complement form
          CLOCK
          JMS CLKINT    /to be XCT'ed
RETURN,   ION           /AC contains address of
                         simulated clock
```

After N/60s of a second, a clock interrupt will occur and control will be transferred to the user subroutine CLKINT with interrupts off. This subroutine will be operating at interrupt time so that it must do all its processing with interrupts off and return to the interrupt dispatcher as soon as possible. Thus, all it can do is set some flags for the user, INSERT a task, or set another clock. Up to four simulated clocks can be set at any one time.

The address returned in the accumulator by CLOCK is the address of the simulated clock. This location contains the twos complement of the number of 1/60s of a second before the clock runs out. This number may be changed at any time before the clock is set to go off, and in particular it may be set to zero, which will disable the simulated clock should the program decide that the interrupt is no longer needed. When the interrupt-time routine is called, the simulated clock has gone to zero so that it is no longer active. If the interrupt-time routine decides to reset a clock, it may reset the timer which called it rather than make a new call on CLOCK. Warning: once a clock has been set to zero by the user, as above, or has been allowed to remain at zero after it has run out, the address given to that clock should no longer be used since it may be assigned to another program.

Example: The code below will result in the character "A" being typed once per second.

```
            IOF
            LAM     -74+1       /=-60 Decimal
            CLOCK
            JMS     CLKINT      /clock handler
            DAC     CLKLOC      /address of clock
            DELETE              /return to LOCOSS
            .
            .
            .
CLKLOC,     Ø
            .
            .
            .
CLKINT,     Ø
            LAM     -74+1
            DAC  I  CLKLOC      /reset clock for another second
            INSERT              /insert the type task
            TYPE
            EXIT    CLKINT      /return to real clock interrupt
            .
            .                   /handler
            .
TYPE,       Ø                   /task entry point
            LAC  (301)          /ASCII "A"
            TPRA                /type it
            DZM  TYPE           /make reenterable
            DELETE              /return to LOCOSS
```

INTABL

Purpose: To change the interrupt handler for a given interrupt.

Calling Sequence:

```
            LAW INTNO
            INTABL
            JMS HANDLR
RETURN,     ————————
```

HANDLR is the entry to the real-time, interrupts-off subroutine which will clear the associated flag, do the necessary processing, and return to the executive interrupt dispatcher. INTNO is the number of the interrupt as given in the following table. The numbers below are octal.

Interrupt Number	Device
1	Dataphone Transmit
2	Light Pen
3	Vertical Edge Flag
4	Horizontal Edge Flag
5	Internal Stop
6	External Stop
7	Dataphone Receive
10	Manual Interrupt
11	Push Button Hit
12	Tape Reader
13	Tape Punch
14	Keyboard
15	Teleprinter
17	Clock

LOAD

Purpose: To load a file from the 1800 disk and start it executing, or return to the user, depending on the contents of the file.

Calling Sequence:

```
AC = file number        /ION or IOF
LOAD
ERROR RETURN            /ION
SUCCESS RETURN          /ION
```

The PDP-7 support routine must be initialized in 1800 core for this command to have any effect. All input files should be closed before calling LOAD.

If the loading procedure produces an error of some sort (e.g., 1800 dead, snarked bootstrap specification, or no such file) control will be passed to the error return with the AC containing the disk error number.

The file addressed is assumed to be in core image format with a START or PAUSE control block at the end of the file.* If the block is PAUSE, control will be returned to the success return with interrupts on. If the block is START, control will be passed to the location addressed in the START block with interrupts off.

RDOCT

Purpose: To read and translate a stream of ASCII characters into a twos complement, signed octal number.

Calling Sequence:
```
RDOCT
JMS INPUT
ERRET, ————————
NRMRET, ————————
```

INPUT is the entry point to a routine which provides a stream of ASCII characters. For example, this call may be "KBDA," which is an effective JMS. All the characters read from the source stream up to the first octal digit or minus sign are ignored. A minus sign starts a number and may not be separated from the first digit by any other character. If no octal digits or a minus sign are read before a carriage return, control is returned to location ERRET. When a number is terminated (by a non-octal digit), control is returned to location NRMRET with

---

* Brender, R.F., Frantz, D.R., Foy, J.L. Jr., and Schunior, T.W., Specialized System Software for Interacting DEC PDP-7 and IBM 1800, Technical Report 11, Concomp Project, University of Michigan, Ann Arbor, October 1968.

the octal integer in the AC and system variable TRMCHR containing the terminating character. Only the low-order eighteen bits of any number are kept. Note also that for signed quantities to have significance the magnitudes of the numbers must be less than $2^{17}-1 (=131071_{10} = 377777_8)$.

Only one of RDOCT and RDDEC should be executing at the same time since they use common intermediate routines.

RDDEC

Purpose: To read and translate a stream of ASCII characters into a decimal number.

Calling Sequence:
```
RDDEC
JMS INPUT
ERRET, ————————
NRMRET, ————————
```

The action for RDDEC is the same as for RDOCT except that all numbers are decimal.

Only one of RDOCT and RDDEC should be executing at the same time since they use common intermediate routines.

PROCT

Purpose: To translate the contents of the AC into a stream of six octal ASCII characters, unsigned.

Calling Sequence:
```
AC = Number
PROCT
JMS OUTPUT
RETURN, ————————
```

OUTPUT is a routine which will accept the stream of ASCII characters produced, for example, TPRA. The number is treated as unsigned. Leading zeros in the output stream are replaced by the contents of system variable LEDING. LOCOSS initialization sets LEDING to 260 octal, an ASCII zero. Thus LEDING = 240 would produce leading blanks, and LEDING = 0 would produce ASCII NULLs (which are ignored by the teleprinter routines).

Only one of PROCT and PRDEC should be executing at the same time since they use common intermediate routines.

PRDEC

Purpose: To translate the contents of the AC into a stream of six decimal ASCII characters, unsigned.

Calling Sequence:
```
AC = Number
PRDEC
JMS OUTPUT
```

The action for PRDEC is the same as for PROCT except that the number converted is decimal.

Only one of PROCT and PRDEC should be executing at the same time since they use common intermediate routines.

2.5 Input/Output

2.5.1 Introduction

LOCOSS I/O is buffered and overlapped as much as possible and is done under interrupt control, i.e., asynchronously with the user's program. Thus when the user calls an output routine and the routine returns, the program can consider the operation to be completed and continue with something else, even though the operation will not be completed for some time. On input, the tape reader and dataphone are overlapped and will read ahead if the user is slow in taking information from the buffers. Calling the keyboard for input results in an initial wait until an entire line has been entered, but parallel lines of execution can be executing while the task awaiting input is BUSY'ed. The only devices not handled under interrupt control are the character generator display routines and the IBM 1800-PDP7 interface routines which handle I/O on a demand basis (Sections 2.5.2.7 and 2.5.2.8).

Recall that on input operations a buffer may be empty (i.e., no input is available) and that on output, the buffer may be full. Control is not returned to the user when his line of execution encounters one of these blocking conditions until the block is removed and the transfer can be made. For example, a call for input from the high-speed reader will not return until the reader has been started and a character read.

Almost all I/O routines have standard calling sequences.

Input is requested by merely calling the associated routine. Upon return the AC contains the input item. Where meaningful, there may be alphanumeric and binary reads from the same device (e.g., the high-speed reader). Output is performed by loading the AC with the item and callin, the desired routine. Upon return, the operation may be assumed to be completed. Again, there may be binary and alphanumeric writes. For alphanumeric write commands, either one or two characters may be passed to the device at the same time. To achieve this, the low-order 16 bits of the AC are considered as two eight-bit bytes with the high-order byte considered first. If the high-order eight bits are zcro, they are ignored (not passed on to the device). The low-order byte is always passed to the device, regardless of its zeroness. For example, the following sequences produce identical results:

```
LAC  (301)                LAC  (140702)
TPRA                      TPRA
LAC  (302)
TPRA
```

That is, they both result in the letters "AB" being typed. The double character transfer is much more efficient in terms of time.

Note: all devices using ASCII code expect DEC-style ASCII, i.e., the high-order or 200 bit is always set.

Each device has an associated initialization routine which restores the device status and buffers to their original state and assigns the standard system routines to service the device. All initialization routines must be called

with interrupts off. All other I/O routines may be called with interrupts off or on but always return with interrupts on. Exceptions to this are noted in the routine descriptions. If the user wishes to handle the I/O for a given device himself, he may do so by first calling routine INTABL with appropriate parameters. INTABL should also be called if any display interrupts are expected. The user is expected to provide a routine for each display interrupt he enables. Since the keyboard and teleprinter routines are highly interactive, both must be changed if one is changed. This condition also applies to the clock and reader routines.

2.5.2 Device Descriptions

2.5.2.1 Paper Tape Punch

The paper tape punch is the simplest of all of the I/O devices since there are no control characters or special cases to worry about. The only special circumstance that can arise is the out-of-tape condition, and, since this cannot be sensed from the PDP-7, the user must ensure that there is enough tape in the feed box before his program starts punching.

Alphanumeric and binary calls to the punch routines may be interspersed from the user's line of control by merely calling the associated routines.

System variable PCHLOC contains the address of the system buffer control block for the punch. By changing the BCB the knowledgeable user may provide a different,

larger buffer, or may play any other tricks he likes.

PCHI

Purpose: To initialize the device-handling routine for the punch to be the standard routine and to clear the punch output buffer.

Calling Sequence: IOF
PCHI
RETURN,—— /IOF

PCHA

Purpose: To punch the one or two ASCII characters in the accumulator in alphanumeric mode.

Calling Sequence: AC = one or two characters.
PCHA
RETURN,—— /ION

PCHB

Purpose: To punch the contents of the accumulator in binary mode, i.e., such that a RDRB call would return with the same word.

Calling Sequence: AC = 18-bit word.
PCHB
RETURN,—— /ION

See the PDP-7 User's Handbook (pp. 76-77) for the tape format of a binary word.

2.5.2.2 Paper Tape Reader

The high-speed paper tape reader on the PDP-7 is somewhat unique among devices of this sort in that there is no way to sense directly the physical end of tape. To remedy this situation, the reader routines recognize the ASCII XOFF character (023) as the physical end of tape and will halt the tape reader when this character is read. This convention eliminates some of the difficulties encountered in reading spurious characters caused by the skewed end of a tape and thus helps to protect the user. If, however, there is no XOFF near the end of the tape, the tape will run out. By means of a timer arrangement with one of the simulated clocks, this condition will be recognized. When it is, the reader routines will insert an XOFF into the buffer for the user's information and stop the reader. Stopping the reader (i.e., clearing the reader select) thus allows the user to insert a new tape without fear of the tape's running away before it is seated properly. The user then indicates that the new tape is ready by the message to the Command Language Interpreter: "(ESC)HSR".

Of course, the character (023) may be legitimate data and not a device control character. If the user program can detect this condition, it may call subroutine RDRSEL which restarts the reader. A user sitting at the keyboard can also detect this condition and restart the reader via "(ESC)HSR." Since the XOFF is transferred to the user program,

no data are lost. System variable RDR-LOC contains the address of the system buffer control block for the reader.

RDRI

Purpose: To initialize the device-handling for the reader to be the standard routine and to clear the current contents of the reader buffer.

Calling Sequence: IOF
RDRI
RETURN,—— /IOF

RDRA

Purpose: To return in the low-order bits of the AC the next eight-bit alphanumeric character from the reader buffer.

Calling Sequence: RDRA
RETURN,—— /AC = 8-bit character; ION

RDRB

Purpose: To return in the AC the next binary word from the reader buffer.

Calling Sequence: RDRB
RETURN,—— /ION

See the PDP-7 User's Handbook (pp. 76-77) for the tape format of a binary word. The end-of-tape character (023) will cause the reader to stop in binary mode also.

RDRSEL

Purpose: To start the high-speed paper tape reader.

Calling Sequence: IOF

RDRSEL

RETURN,——— /with IOF

RDRSEL issues the actual hardware select instructions to set the reader in motion and sets a timer so that if there is no tape in the reader, the device routines can insert an XOFF into the buffer and clear the select. The CLI command "HSR" calls RDRSEL. The user should not call RDRSEL from either his program or the CLI unless the reader has not been started since the beginning of a program or an XOFF has been encountered (either on the tape or as the result of a time-out).

2.5.2.3 Teleprinter

The teleprinter and associated routines provide a character-oriented output device with some formatting. The system variable "TABCNT," accessed directly in the system communication area, defines equi-sized tab stops on the teleprinter. Since the KSR-33 does not have a real tab mechanism, the tabs are simulated from the output routines by outputting the appropriate number of spaces to position the carriage at the next "stop." TABCNT is defined on the LOCOSS symbol definition tape and may be accessed directly by the user. For example,

```
LAC (12)                /Octal Radix
DAC TABCNT
```

will effect "stops" at carriage positions 10, 20, 30,... (decimal). A value of zero will be defaulted to a value of one. TABCNT may also be set by the CLI command TAB (see 2.3 Command Language Interpreter). Other editing of the output stream is given below.

Since the teleprinter is a common output device for program communication, the situation often arises that two tasks may wish to print a message at the same time. Since the output routines are character-oriented and may be called directly by any task, the result may be unintelligible interleaving of character information from the two tasks. To avoid this difficulty, the user may write his programs to employ the programming convention of "seizing" the use of the teleprinter at the beginning of a print line, and "releasing" it at the last character of the line, the carriage return. Then, during the period that the teleprinter is being used by one task (i.e., while it is "seized"), another task may not seize it, so that a whole line is printed without interference. After the teleprinter has been released, any task may then seize it and print its line of information.

This convention is implemented in LOCOSS by means of the system routine SEZTPR. Thus, whenever a user wishes to start a teleprinter line, he calls SEZTPR as a subtask. Upon return from SEZTPR, the user may print the line without fear of interference; that is, he may make successive calls on TPRA until he prints a carriage return. Calling TPRA with a carriage return as the low-order byte in the AC automatically releases the teleprinter so that another task may seize it.

Of course, if a user does not expect to be printing from independent tasks he need not bother calling SEZTPR at the beginning of a line, since no other task will have seized the teleprinter and be in the process of printing.

TPRI

Purpose: To initialize the device-handling routine for the teleprinter to be standard routine, to clear the contents of all print buffers, and to release the teleprinter if it has been seized.

Calling Sequence:

```
IOF
TPRI
RETURN,——          /IOF
```

TPRA

Purpose: To print the one or two ASCII characters in the AC.

Calling Sequence: AC = One or two ASCII characters.

```
        TPRA
RETURN,———              /ION
```

The following translations of the ASCII character set are made in the stream of output characters:

1) Carriage return (215) is mapped into a carriage return and line feed (212). (Note that (015) is typed alone as only a carriage return.)
2) Line feed (212) is not printed. (Note, however, that (012) will print as a line feed.)
3) Zero characters are not printed, i.e., they are ignored.
4) The tab character (211) generates enough spaces to move the carriage to the next tab stop, i.e., next multiple of TABCNT.

A carriage return in the low-order byte of the accumulator will "release" the teleprinter (see SEZTPR).

SEZTPR

Purpose: To seize the use of the teleprinter routines for the duration of a print line.

Calling Sequence:

```
SUBTSK          /IOF or ION
SEZTPR
RETURN,——       /ION, with AC
                  restored
```

If the teleprinter routines have been seized and not released by another task, control will not be returned to the calling task until the release is given. Thus when control is transferred to location RETURN, the teleprinter routines may be called without fear of interference. The accumulator is saved and restored around the call to SEZTPR. A call to TRPA with a carriage return in the low-order byte of the accumulator will release the teleprinter. If any task expects to use the seize-release convention, all tasks using the teleprinter should do so.

2.5.2.4 Keyboard

The teletype keyboard is treated as a line-oriented input device. That is, a call for input is essentially a call for the input of an entire line—a string of characters ended by a carriage return. The keyboard routines will collect the line and, upon receipt of a carriage return, will feed characters one at a time to the user via successive calls on KBDA. During collection of the line, and at any time until the input of a carriage return, the line may be altered by the two editing characters, LEFTARROW (←) and RUBOUT. LEFTARROW causes the last-typed character to be deleted from the line image. Two LEFTARROWs delete the last two characters, etc. Backspacing beyond the first character in a line has no effect. The ESC character (control-shift-K) cannot be deleted by LEFTARROW. Typing the RUBOUT character causes

deletion of the whole line, at which time a pound sign (#) and a carriage return are then printed by the keyboard routines so that input starts on a new line. (The output of "#" and other system-control characters depends on the current state of the echo condition. (See ECHO below.)

If, when a carriage return is entered, the first character in the edited line is ESC, the line is sent to the CLI. When the CLI is finished, the keyboard is again available for input (see 2.3 Command Language Interpreter).

If, at any time, the input buffer is full when a character is typed, a LEFTARROW and RETURN will be typed back at the user indicating that the last-typed character did not enter the buffer (see ECHO). The user must then delete at least one more character so that he may enter a carriage return to end the line. The input buffer has room for seventy-two characters; that is, seventy-one characters plus carriage return.

When no read is outstanding on the keyboard (i.e., when KBDA has not been called, or when a line has been entered and not entirely read from the buffer by repeated calls on KBDA) the only input allowed is a command line (i.e., a line begun with ESC). Any other input is ignored. In fact, a "#(RETURN)" is immediately typed back at the user if he tries to type anything else (see ECHO).

So that a user sitting at the teletype may know when a program is expecting input, the program may choose the option of supplying a prompting character (question mark-?) when a new line is expected. This choice is given in the system variable PRMPTC. When PRMPTC is zero, the prompting character is given. When PRMPTC is nonzero, the prompt is not given. PRMPTC is defined in the LOCOSS symbols and may be accessed directly by the user. For example,

```
CLC
DAC     PRMPTC
```

will turn prompting off. Deletion of the input line by RUBOUT or intervening CLI commands will cause the prompting character to be re-issued at the beginning of the next line. System reinitialization sets prompting on.

If a user starts an input line while a program is printing, the printing will be halted until the input line has been finished by either a carriage return or RUBOUT.

ECHO

The teletype on the PDP-7 can operate in either half- or full-duplex mode by means of a throw switch on the PDP-7. Half-duplex operation means that hardware in the teletype controller will automatically print any character that is struck on the keyboard. Full-duplex means that

printing is completely independent of keyboard operations. Normal operation under LOCOSS expects the switch to be in the full-duplex position.

The two modes of operation for control of the keyboard are designated as "echo-on" and "echo-off." Which of these two modes the keyboard routines should assume is determined by the state of the system variable ECHOSW. When ECHOSW is zero, the mode is echo-on; when ECHOSW is nonzero, the mode is echo-off. For example, "DZM ECHOSW" will change the mode to echo-on. The mode may also be changed by the ECN (echo-on) and ECF (echo-off) commands to the CLI. System reinitialization sets echo-on.

When the system is in echo-on state, the keyboard routines will print back at the user any character struck on the keyboard and additional control characters to help format the input and thus make it more readable. Specifically, when in echo-on mode the keyboard routines will:

1) Print any character struck on the keyboard;
2) Supply a line feed after every carriage return typed by the user;
3) Supply a suitable number of spaces to position the carriage at the next tab stop when a TAB is typed by the user (see 2.5.2.3 Teleprinter, variable TABCNT);

4) Print an "AT" sign (@) when an ESC character (control-shift-k) is typed by the user; and
5) Supply a LEFTARROW and carriage return on buffer overflow, and a pound sign and carriage return on RUBOUT or unsolicited input.

The echoed characters are kept in a separate print buffer. If this buffer overflows, no characters which are struck will be printed or entered into the keyboard buffer until the echo buffer is emptied. In any case, the printed line will be a faithful reflection of the contents of the keyboard buffer.

In echo-off mode the system will not print the character struck nor will it supply any of the control characters described above. Thus, "echo-off" means that nothing will appear on the teleprinter except what is specifically printed by the user's program or the Command Language Interpreter.

Echo-off mode is usually used in conjunction with the 337 display, so that the input line is displayed on the screen rather than on the teleprinter. To assist this process, the system keyboard input buffer is kept in a state suitable to be used as a 337 display file of the currently edited input line. This display file assumes

it will be entered via a PJMP and that the character generator is in the seven-bit mode. The structure of this file is:

```
KBDBUF,          EDS!CHR
                   --  }
                   --  }
                       }  /edited line
                   0   }
                   .   }
                   .   }  /unused part of buffer
                   .   }  /are null characters
                   0   }
                   1      /escape character
                 EDS!SVEC
                 4C45     /put up the cursor
                 POP
```

The system variable KBDLOC contains the address of the buffer control block for the system input buffer. (See 2.6 Buffer Management Routines for the structure of the buffer control block.) So, in order to insert the editing line into a display file, the following coding is sufficient:

```
LAC  I  KBDLOC   /pick up the keyboard buffer...
                 /...location, i.e., KBDBUF
DAC     EDILIN   /store it in the display file
```

where EDILIN is the second word of a PJMP instruction in the user's display file. The user-supplied character generator table is assumed to dispatch on the ASCII character set, with zero a null character and ore (1) the generator escape character (see Section 2.5.2.7 for a use of this facility).

KBDI

Purpose: To initialize the device-handling routine for the keyboard to be the standard routine, clear all input buffers, and reinitialize the Command Language Interpreter.

Calling Sequence: IOF
KBDI
RETURN, —— /IOF

KBDA

Purpose: To return in the low-order bits of the accumulator the next eight-bit character from the input buffer, and also to cause the prompting character to be given if prompting is on and the buffer is empty.

Calling Sequence: KBDA
RETURN, —— /ION

See the discussion under KEYBOARD for the effect of system variables ECHOSW and PRMPTC, and input editing.

2.5.2.5 IBM 1800 Logical Disk Files*

The logical file system for the IBM 1800 is available to the PDP-7 across the 1800-7 interface when the appropriate interrupt service subroutine has been initialized in 1800

* Brender, R.F., Frantz, D.R., Foy, J.L. Jr., and Schunior, T.W., Specialized System Software for Interacting DEC PDP-7 and IBM 1800, Technical Report 11, Concomp Project, University of Michigan, Ann Arbor, October 1968.

memory. Details on the file routines are available in another memo, but the features available to the PDP-7 are briefly discussed below. In general, the user can treat a disk file as a serial source and sink for characters. He can create, open, write into or read from, close, and destroy files.

To write a file, the user must first create the file (if it does not already exist), open it for output, write into it, and finally close it. To read the file, he must open it for input, read from it, and close it. By convention, symbolic files reside on the disk in EBCDIC and are translated to and from ASCII on reads and writes by the IBM 1800 interrupt service subroutine.

With regard to the file routines, the interaction between the two machines is master (PDP-7) to slave (IBM 1800). The PDP-7 tells the 1800 when it wants something done, the 1800 tries to do it, and then tells the PDP-7 whether the request was successful. Thus, every transfer and command is acknowledged. If the function requested cannot be carried out (e.g., the file to be opened does not exist or there is an end-of-file on a read), an error indication is returned to the PDP-7 and the LOCOSS disk routines will transfer to a user error routine. The disk routines operate with interrupts on or off and do not change the interrupts-enabled status.

DSKGET

Purpose: To get an eight-bit character from the currently opened input file.

Calling Sequence: DSKGET

RETURN, ———

The character returned will be translated from EBCDIC to ASCII depending on the manner in which the file was opened. (See below.) If no file is open, or if translation to ASCII was requested and is not possible, or if the end of the file has been reached, the 1800 will give an error indication and control will pass to the current error routine.

DSKPUT

Purpose: To put the one or two eight-bit characters in the accumulator into the currently opened output file.

Calling Sequence: AC = 1 or 2 characters

DSKPUT

RETURN, ———

The characters will be translated from ASCII to EBCDIC or not depending on the way the output file was opened. (See below.) If no file is open or if translation to EBCDIC was requested and is not possible, or if there is no more room on the disk for files, the 1800 will give an error indication and control will pass to the current error routine.

DISK

Purpose: General purpose communication with the IBM 1800 service routine.

Calling Sequence: AC = 8-bit character or file number (if applicable)

DISK

Disk op code

RETURN, ——— AC = character (if applicable)

The disk op codes are given below in symbolic form as they are defined on the LOCOSS symbol tape. They represent functions which the IBM 1800 interrupt service routine will perform. This list may be extended. Only the low-order eight bits of the accumulator are used. If any error occurs, control will be passed to the current error routine. A typical call would be:

```
        LAW 15
        DISK   /open file 15 for input, translated.
        D.OPIT
      RETURN,——
```

SYDSK

Purpose: General purpose communication with the IBM 1800 service routine with an immediate error return.

Calling Sequence: AC = eight-bit character or file number (if applicable)

```
                SYSDSK
                Disk op Code
        ERRET,  AC = error number
        NRMRET, AC = character (if applicable)
```

S.SDSK is identical to DISK except that if an error occurs, control will be transferred to location ERRET instead of the error routine set up through DSKERR. If no error occurs control is transferred to location NRMRET. DSKERR's error exit is not disturbed by a SYSDSK call.

DSKERR

Purpose: To set the disk error return to a user program.

Calling Sequence:

```
        DSKERR
        ERROR INSTRUCTION
RETURN, ———
```

The error instruction will typically be a JMS or JMP to the user's program. The last such error routine set up by calling DSKEkR will be executed by the LOCOSS disk error handler if a disk error occurs in subroutines DSKGET, DSKPUT, DISK, ICYRED, or ICYWRI. Upon entry to the user's error routine, the accumulator will contain the error number (see below) and location 20 (octal) will contain the service request as presented to the 1800; i.e., bits 2-9 contain the operation code and bits 10-17 contain the argument.

If the user's error routine was entered by a JMS and it does a normal subroutine exit to the LOCOSS disk error handler, the entire accumulator will be returned to the user disk call which caused the fault, as if nothing had intervened. The default handling for a disk error (restored by RESET and in effect until the user calls DSKERR) is to print an error message (error number 3) and DELETE the current line of execution.

The error codes are below (in octal). Some of these error codes may also appear if the 1800 file routines are snarked, but in that case it's all over anyway.

Table 2. Disk Operation Codes

OP CODE	FUNCTION
D.ECHO	Return the character in the AC. (Used mainly for determining that the 1800 is awake.)
D.ECOA	Return the ASCII equivalent of the EBCDIC character in the call.
D.ECOB	Return the EBCDIC equivalent of the ASCII character in the call.
D.CRE	Create a file with the number given in the call.
D.DES	Destroy the file with the number given in the call.
D.OPOU	Open the given file for untranslated output.
D.OPOT	Open the given file for translated output (i.e., translate all characters to EBCDIC before storing on the disk).
D.PUT	Put the character in the AC into the currently opened output file.
D.CLO	Close the output file.
D.OPIU	Open the given file for untranslated input.
D.OPIT	Open the given file for translated input (i.e., translate the EBCDIC characters to ASCII before returning).
D.GET	Get a character from the currently opened input file.
D.CLI	Close the input file.

Table 2 cont'd

OP CODE	FUNCTION
D.CLR	Close any files that are opened (cannot fail).
R.REDT	Get a character from the IBM 1800 copy task translated.
D.WRIT	Give a character to the IBM 1800 copy task translated
D.REDU	Get a character from the IBM 1800 copy task untranslated.
D.WRIU	Give a character to the IBM 1800 copy task untranslated.

Table 3. Disk Error Codes

CODE (octal)	MEANING
1	Invalid command.
2	"Open" request rejected—no such file.
3	"Open" request rejected—file already in use by another program at the moment.
4	"Put" error—file not open.
5	File overflow—no more room on disk for "Put."
6	"Get" error—file not open.
7	"Close" error—file not open.
10	"Create" error—file already exists or number not in legal limits.
11	"Create" error—no more room on the disk.
12	"Destroy" error—no such file exists, or the file is open. It must be closed before being destroyed.
13	Translation to ASCII not possible on read from file or translate command.
14	Translation to EBCDIC not possible. If encountered on a "Put," a percent sign (%) is inserted in the file.

Table 3. cont'd

CODE (octal)	MEANING
15	Copy port on IBM 1800 (ICY) busy. Try again later.
16	Illegal use of bootstrap command (on a file numbered more than 12 octal).
357	End-of-file read on a "Get," i.e., the last character has already been read. The file is closed after this error code is returned.
374	The 1800 copy port was quiescent on a WRITE.
375	The 1800 copy port was quiescent on a READ.
376	The 1800 didn't take the service request.
377	The 1800 took the service request but didn't come back with an answer.

2.5.2.6 IBM 1800 Copy Port

Another feature of the PDP7-IBM 1800 interface routines is a logical connection to the copy port of the IBM 1800. Thus the user may use other 1800 peripherals than the disk in moving data to or from the PDP-7 by issuing to the IBM utility program the proper copy to or from "PDP-7." Running programs in the 1800 also have access to this port.

PDP-7 access to this facility is available through the "DISK" subroutine with operation code parameters D.REDT and D.WRIT for translated transfers (i.e., symbolic information) and D.REDU and D.WRIU for untranslated information (i.e., binary characters). There is, however, a special error code (octal 15) related to these commands which indicates that the 1800 copy program is not ready to service the call; e.g. data from previous D.WRITs is still being copied to the typewriter, or the card reader hasn't finished reading a card in response to a "D.REDT." This "not ready" response indicates only a temporary blocking condition and that the same command may meet with success if attempted later. The four ICY (IBM Copy) routines provided recognize this "not ready" condition and try again after a decent interval. (The interval is to allow the 1800 to exit from its interrupt servicing status and do the other I/O work required.)

If the 1800 returns "not ready," these routines will take BUSY exits, allowing other task processing to continue. Upon return from BUSY, the operation will be retried unless a given time (about fifteen seconds) has passed since the first try. If fifteen seconds have not passed, another BUSY exit will be taken and the whole process repeated. If the time is up, it will be assumed that the 1800 really isn't trying very hard, and control will be passed to the current disk error routine with an error number of 374 (octal) for WRITE and 375 for READ. Any other errors will also cause control to be transferred to the current disk error routine.

ICYRDT

Purpose: To read an eight-bit ASCII chracter from the IBM 1800 copy port.

Calling Sequence:

```
ICYRDT          /IOF or ION
RETURN,——       /ION
```

If EBCDIC to ASCII translation of the character is not possible or if the 1800 copy port appears to be quiescent, i.e., never comes ready, control will be passed to the current disk error routine. (See DSKERR.)

ICYWRT

Purpose: To write the one or two eight-bit ASCII characters in the accumulator to the IBM 1800 copy port.

Calling Sequence: AC = one or two characters /ION or IOF
ICYWRT
RETURN,—— /ION

If ASCII to EBCDIC translation of one of the characters is not possible or if the 1800 copy port appears to be quiescent, i.e., never comes ready, control will be passed to the current disk error routine. (See DSKERR.)

ICYRDU

Purpose: To read an eight-bit binary character from the IBM 1800 copy port.

Calling Sequence: ICYRDU /IOF or ION
RETURN,—— /ION

If the 1800 copy port appears to be quiescent, control will be passed to the current disk error routine.

ICYWRU

Purpose: To write the one or two eight-bit binary characters in the accumulator to the IBM 1800 copy port.

Calling Sequence: AC = one or two characters /ION or IOF

ICYWRU

RETURN,——

If the 1800 copy port appears to be quiescent, control will be passed to the current disk error routine. Recall that if the high-order byte in a two-character transfer is zero, it will be ignored.

2.5.2.7 Character Generator Package

The character generator package allows the user to display character data on the face of the screen by treating the display as sink for ASCII characters in a format compatible with other ASCII-oriented devices.

The package is available as a set of routines to be loaded from the IBM 2310 disk (see system routine LOAD and the system file directory) or from the tape labeled "HIGH CORE CHARACTER ROUTINES." Since there are no relocation and program linkage facilities in the current system, the package is loaded into a predesignated area: 16000-17577 octal. The dispatch table and display file for the characters occupy (approximately) locations 16000-17000, and the file-handling routines occupy about 17000-17440; locations 17440-17577 are used as a default file location.

The file-handling routines and their parameters are available to the user through a local transfer vector at location 17000. In general, for the case of default parameters, entry DISPLA will accept one or two ASCII

characters (in normal I/O format) and add them to the file. If more than seventy-two (decimal) characters are put on a line, a hyphen and a carriage return-line feed sequence is automatically inserted after the seventy-second, starting a new line. A maximum of forty lines is displayed, and the character file buffer is circular so that only the last forty lines generated are on the screen, with the oldest line being deleted from the top as new ones are added on the bottom.

The package's display file also provides an echo line on the display face for input from the keyboard. A cursor appears at the bottom of the screen to indicate the entry point for the next character to be typed. This echo line is a faithful reproduction of the input line being collected by LOCOSS, reflecting all editing. No more than seventy-two characters will be accepted before a carriage return; and if LOCOSS is not expecting, or will not allow, input, nothing will appear on the echo line even if something is typed (see 2.5.2.4 Keyboard). The echo line is quite useful in conjunction with keyboard echo-off mode. There is also a facility for the user to insert a push-jump into the display regeneration loop so that he may display other than character information on the face of the screen.

If the user wants to use only the characters and not the file routines, he may load the package and then use the file

routine area (17000-17577) for his own uses. Then, in his display initialization sequence, he should include the following to initialize the character generator:

```
        LAW 216                 /seven-bit mode, base of
                                /dispatch
                                /table at 16000
        SCG
```

After this, to display characters the user merely includes the enter data-state instruction and escape character around an ASCII list; for example to display "AB" :

```
        EDSICHARR               /=1151
        301                     /=ASCII "A"
        302                     /=ASCII "B"
        ESCCHR                  /escape character.
```

(See below for control character definitions.)

Character File. The display file created by the package is highly parameterized. The following is a summary of the location in the transfer vector which will be referred to symbolically in the discussion below. All numbers are octal unless otherwise specified.

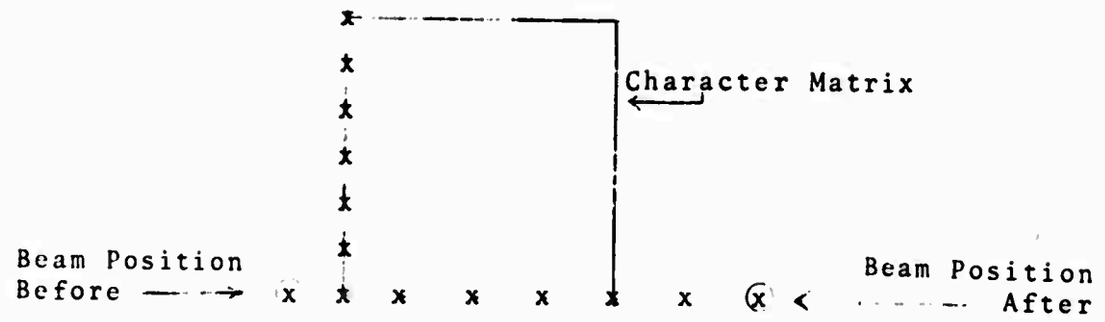
NAME	LOCATION	DEFAULT	DESCRIPTION
INDISP	17000		Entry to initialize display and file.
STOPDY	17002		Entry to stop display.
RESTRT	17004		Entry to restart display.
DISPLA	17006		Entry to add characters to file.
DSPBUF	17010	17454	Beginning of character buffer.
BUFSIZ	17011	123	Size of buffer.
PDLIST	17012	4370	Location of push-down list.
JMPLOC	17013		Location for a PJMP to the user's file.
YLOC	17014	1760	Vertical position on screen for display file.
LINES	17015	-50	-Maximum number of lines to be displayed.
CHARLIN	17016	-111	-Maximum number of characters or a line.
PBUSEF	17017	0	Push-button use flag.
SICLOC	17020	4	Initial conditions for the display.

Characters. The complete set of ASCII characters is programmed for the character generator with a dispatch table operating in the seven-bit mode. This set includes all printing graphics and the following special cases for non-printing combinations.

OCTAL	ASCII	EFFECT
000	NULL	Does nothing.
001	SOH	Escape from character mode.
004	EOT	Down arrow (↓). (For editor usage.)
005	WRU	Diamond (◇). (For editor usage.)
011	TAB	Eight spaces.
012	LF	Line feed. Decrease y-coordinate by 12 (decimal).
014	FORM FEED	Null. (For editor usage.)
015	CR	Carriage Return. Set x-coordinate to zero
021	ETB	Null. (For MTS use in ending a prompt line.)
023	XOFF	Null. (For editor usage.)
033	ESC	Displayed as "@" (LOCOSS Command Character.)
161		One space.
162		Two spaces.
.		.
.		.
.		.
177		15 spaces. (161 to 177 are for TAB spacing).

The remaining nonprinting characters are displayed as a single space.

These characters are displayed in a 7 x 7 raster-point matrix with the graphic itself on a 5 x 7 matrix.



Readable, solid characters (1/4 inch high) are produced at scale two, at which scale 73 characters can be displayed across the screen, and 42 lines can be displayed down the screen.

INDISP

Purpose: To initialize the display and the display file according the current state of the parameter list and physically to start the display.

Calling Sequence: JMS INDISP or INSERT or SUBTSK INDISP INDISP

Location DSPBUF is expected to contain the first location of a region of core which is to be used to keep the circular file of characters displayed on the screen. BUFSIZ contains the size of this buffer. The minimum size region which can be used is ten (decimal) plus the maximum number of characters allowed on the line. For example, if the maximum number of characters per line were 73 decimal (i.e., parameter location CHARLN contained -73) then the minimum number that should appear in BUFSIZ is 83. The

default values for these three parameters (i.e., the values occupying these locations when the package is first loaded) provide a small display file in the upper part of core, just below the system loader.

The contents of location PDLIST will be used to set the push-down pointer for the display. The default condition is to set it at 4370, by agreement with LOCOSS, so that four levels of push-down are provided before normal task space (at 4400) is invaded. The contents of location SICLOC will be used to set the initial conditions in the display. The default value of 4 provides for no edge flags, a one-by-one sector, and no light pen, edge flag, push-button or internal stop interrupts. The low-order ten bits of location YLOC contain the vertical coordinate to be used in positioning the display file. The default value of 1760 starts the file at the very top of the screen; a value of 1000 would start the display at about the middle of the screen. Location LINES contains the twos complement of the maximum number of lines to display. Thus the file will contain either the maximum number of characters [C(BUFSIZ)-10], or lines [-C(LINES)], whichever is smaller. The default value of 40 (decimal) lines just about fills the screen.

Location PBUSEF controls the use of the push buttons. If it contains a nonzero number, the character generator package will neither disturb nor interpret the push buttons.

If it contains zero, the package will interpret the push buttons as explained in routine DISPLA, and the initialization routine will clear buttons 6-11. The default value is for the package to use the buttons.

There is a two-word no-op (i.e., zeros) in the display file created by INDISP. Location JMPUSE contains the address of the first of these two locations. If a user wants to add his display file to the display file maintained by the package he may replace the two words addressed by JMPUSE with a PJMP to his display file (being careful about core bank bits, of course). The file so added should then end with a POP. Any change of this sort requires that the display not be running at the time of the change. (See STOPDY and REDISP.)

INDISP sets the LOCOSS interrupt handling for external stop and manual interrupt. The external stop handling is necessary for the STOPDY subroutine which is called whenever a character is added to the file. Thus the user should handle all external stops through STOPDY. The default handling for manual interrupt is to reinitialize the display file, i.e., call INDISP. If the user wishes to use this button for other reasons he need only call INTABL after INDISP to reset the handling.

After initialization of the display file, INDISP physically starts the display by calling RESTART.

STOPDY

Purpose: To stop the display if it is going.

Calling Sequences: JMS STOPDY or INSERT or SUBTSK.

STOPDY STOPDY

STOPDY expects that the LOCOSS interrupt handling for external stop has been set up already; thus, INDISP should be called before STOPDY.

Since there is no way to determine whether the display is running or not, the following scheme is used to ensure that it is indeed stopped when control is returned to the user. The routine issues the display stop IOT (STPD), waits for the external stop flag to appear, clears the flag if it does, and then returns immediately, If, however, the display was not running at the time the stop command was issued, no flag will appear; STOPDY will wait approximately 1.5 milliseconds (which is the maximum time that the display will delay response to the stop command) and then return to the user, who is thus assured that the display is not running.

RESTRT

Purpose: To physically restart the display (on the character file regeneration loop).

Calling Sequences: JMS RESTRT or INSERT or SUBTSK
RESTRT RESTRT

RESTRT may be called after STOPDY to restart the display. It changes nothing in the file but merely issues the five IOTs necessary to get the display started at the beginning of the file. The address of the push-down list initialization is taken from locations PDLIST, and the initial

conditions are taken from location SICLOC.

DISPLA

Purpose: To add the one or two characters in the AC to the display file.

Calling Sequences: AC = one or two characters, then...

JMS DISPLA or INSERT or SUBTSK

DISPLA DISPLA

The character generator dispatches on the low-order seven bits of an ASCII character, but in the discussion of control characters below, the high-order bit (i.e., bit "200") is expected to be set for the character to be recognized as given control character. The input stream of characters is edited in the following manner. If the character is null (000), rubout (377), character mode escape (001 or 201), or line feed (212), it is ignored. If the character is a carriage return (215), a line feed is added. If the character is a TAB (211), the extra spacing required to postion the beam at the next "tab stop" is calculated and inserted into the file. The tab stop spacing is calculated exactly as in teleprinter output so that a line output to the either teleprinter or the display would have the same appearance in spacing on both devices. In particular, the tab spacing is set from system variable TABCNT. (Note, however, that a tab on the input echo line gives a constant eight spaces and not the proper tab spacing. This sad state is caused by the fact

that the echo line display file is the input buffer itself and cannot be changed for mere display purposes.)

If there is already the maximum number of characters on a line (given by the contents of the parameter location LINCNT, as a negative number in twos complement form) and the input character is not a carriage return (215), a hyphen and carriage return-line feed sequence are added to the file so that the current character then will appear on a new line. When there are more than the maximum number of lines in the file (given by the contents of parameter location LINES, as a negative number in twos complement form), the top line of the file is deleted.

When a carriage return is encountered, the controls of parameter location PBUSEF are checked. If PBUSEF is non-zero, return to the user is immediate. If PBUSEF is zero, the push buttons are used as a means of slowing down or stopping the addition of lines to the file, i.e., the return to the user after addition of a carriage return is slowed down or stopped. Specifically, return will not be made to the user while push-button ten is on. If push-button eleven is on, push-buttons 0-5 are read as a binary number, one (1) is added to them, and the resulting sum is used as the number of 1/60s of a second to delay return to the user. Thus push-buttons 0-5 all off provide 1/60 of a second delay between lines, and push buttons 0-5 all on provide 64/60 or 1.07 seconds delay between lines. Both methods of delaying

return employ a BUSY exit so that other task processing may take place in the meantime.

2.5.2.8 Dataphone

The 201A dataphone, associated interface, and software provide a half-duplex source and sink of 8-bit characters from and to the regular telephone network. They provide a medium-speed (2000-bits-per-second) link to any other data-set which is operating under identical message formats. In particular, this includes the Data Concentrator at the University of Michigan Computing Center, which in turn provides access to the IBM 360/67.

The dataphone package is available as a set of routines to be loaded from the IBM 2310 disk (see system routine LOAD and the system file directory) or from the tape labeled "DATAPHONE PACKAGE." Since there are no relocation and program linkage facilities in the current system, the package is loaded into a predesignated area: 14400-15777 octal. This choice of addresses allows the character package to be loaded also if so desired.

All links to the dataphone routines are made through the LOCOSS transfer vector. Before the package is loaded, these transfer vector locations will contain the address of the DUMMY routine so that calls can be made to the dataphone as if the routines were actually in core, and all programming considerations (except perhaps timing)

will be the same as when the dataphone package is actually present.

The following is a brief description of the problems involved in handling this device so that the user can appreciate the organization involved and the calling conventions chosen.

The nature of the transmission is serial (bit-by-bit, rather than character-by-character), synchronous (both ends must agree on when a message is being transmitted, and transmission takes place at a constant rate), and half-duplex (transmission proceeds in one direction at a time). These three features, and the fact that the line uses standard voice-grade telephone lines which are not of impeccable quality, have led to rather elaborate hardware and software conventions in message formatting and error checking. Messages are checksummed linearly and every data message is acknowledged (i.e., a message is sent to the transmitting computer) if its received and calculated checksums agree, or given a negative acknowledgment if they do not. The latter case implies a request to retransmit the message since the information received could not be trusted. In addition, some eight-bit patterns must be used as line control characters (start of message, end of message, etc.), so that conventions must be followed to allow transmissions of binary information, since a binary combination in the data stream may not be meant as a control character.

Most of this mechanism is removed from the user and is reflected only slightly in the basic output routines. This fact makes the basic dataphone calling sequences device-dependent (i.e., not exactly like the other devices), but rather simple routines for the special case of alphanumeric data have been written to resolve this conflict For this special case of all-ASCII transmissions, message boundaries are cued on a special character (carriage return). Since the device is also intended to provide binary transmission, this resolution is difficult in general, so that other specific handling is left to the user, and the lowest level handling will be of eight-bit, transparent binary characters.

The system routines handle all other message formatting, error checking, requests for retransmission, etc. Incoming messages are not passed to the user when he requests input until the checksum is verified; and no additional messages are sent until the last has been positively acknowledged. The only exception to this rule occurs when an "attention" is generated from either end. An attention clears any blocking condition due to bad messages or unacknowledged transmissions, and, when sent to MTS (Michigan Terminal System; see Bibliography) will cause the program executing in the 360 to be interrupted.

Dataphone Printouts. Messages which have bad checksums are considered rare, and so a comment is printed on the teleprinter so that this special condition can be noted (for possible diagnostic purposes) although the retransmission is handled automatically. If a message is received with a bad checksum the comment "%@R" is printed, and if a negative acknowledgement is received in response to a bad transmission from the PDP-7 the comment "%@T" is printed. If a spurious positive or negative acknowledgment is received (i.e., the PDP-7 isn't expecting one), something is amiss and for lack of anything better to do, the comment "%@S" is printed. An attention received at the PDP-7 will clear up any blocking condition and cause the message "%@A" to be printed on the teleprinter. A summary of all control characters, including ACK and NACK, is given below.

Any message received at the PDP-7 which is started by an ASCII even-parity STX (202) will have the STX stripped and the rest of the message sent to the user. A message which starts with an even-parity SOH (201) will be printed on the teleprinter preceded by "%" instead of being directed to the user. A message to the PDP-7 which needs an SOH at the beginning as part of the data and is not intended to be printed should begin with an STX to avoid the SOH effect. The STX will be stripped by the device handler and the SOH will then appear first to the user. Thus, to be on the safe side, the user should start every

transmission to the PDP-7 with an STX. The STX will not be given to the user routine in the PDP-7. (Note of warning: other computers along the line also use the initial STX/SOH convention as a text/command control character, stripping it off and dispatching the rest of the message either to the user or to its own command language interpreter, respectively. Thus, an additional STX must be added to the message for every computer or device controller in the line of transmission which adheres to this convention. Most notably (and infamously), this includes the Data Concentrator and the 360 support routines. For example, to send an initial data SOH from an MTS program to the PDP-7, the following sequence must be sent: STX STX STX SOH. This is a strange convention, and may be changed in the future.

Calling Conventions. In view of the foregoing, the following basic calling conventions have been adopted. For reading from the dataphone, the user merely calls the read routine, DFNRED. All message checks and requests for retransmission are handled by the low-level routines, and the user is completely insulated from them. An incoming message beginning with an SOH will be printed on the teleprinter preceded by "%" and a message beginning with STX will have the STX stripped and the rest of the message sent to the user. The end of the message is marked by one of the end-of-record characters: ETX, ETB, EOM, or EOT. The entire list is included here for completeness' sake although, except for extraordinary (probably pathological) cases, only ETX and ETB will be received from other machines. The user should always attempt to read to the end of a message, for it is only then that the local system will transmit a positive acknowledgment to the other machine so that the latter can send another message.

In transmission of messages over the dataphone, the user must do three things: request the dataphone transmission facilities, send the message, and indicate the end of message. In requesting permission to transmit (via a SUBTSK

of task DFNREQ) the user implicitly waits until previous dataphone transmissions have been completed and acknowledged, and once permission is granted (return from DFNREQ), he is ensured that no other task will be allowed at the dataphone until the current message is finished. Once the user has been assigned use of the dataphone, he may call subroutine DFNSND and send one or two characters at a time (in a format similar to the other device transfers). The user indicates end-of-message by calling DFNSND with bit zero of the accumulator set. This call allows him to specify the default end-of-record character (ETX) or one of his own choosing.

For the case of all ASCII, alphanumeric transmission (e.g.,communication with MTS as a glorified teletype terminal), there are two routines which have calling conventions similar to the other I/O devices and which make the mappings from the record-oriented dataphone to a stream of characters. On input, DFNRA maps the end-of-text character into a carriage return. On output to DFNSA, a carriage return ends a line and signals that the next character to be sent starts a new record. Both DFNRA and DFNSA call the respective lower-level routines described above so that the user should be consistent in his choice of input or output routines.

To send an "attention" over the dataphone the user should call subroutine DFNATN. Attentions clear up any blocking conditions existing at either end.

DFNI

Purpose: To initialize device handling for the dataphone to be the standard routines, to clear the dataphone buffers, and initialize the hardware so that dataphone operation may begin.

Calling Sequence:
```
IOF
DFNI
RETURN,——
```

Subroutine DFNI initializes the hardware by clearing all status bits and then setting "TERMINAL READY," and setting the transmission frame size to eight-bit bytes. DFNI should be called (from the program or the CLI command "DAT") immediately after loading the dataphone package.

DFNRED

Purpose: To obtain the next eight-bit character from the dataphone receive buffer.

Calling Sequence:
```
DFNRED          /IOF or ION
RETURN,——       /with character in
                /AC,ION
```

The end-of-record character (i.e., the last character in the buffer) is one of the following: ETX, ETB, EOM, or EOT, and is transferred to the user.

DFNREQ

Purpose: To obtain the use of the dataphone and to start transmission.

Calling Sequence: SUBTSK
DFNREQ
RETURN,——

Return is made when the dataphone is free for transmission, i.e., when all previous messages have been sent and positively acknowledged. Upon return the user is free to call subroutines DFNSND. DFNREQ must be called before every record is sent, and a DFNSND call with bit zero of the AC set must be made to end every record. DFNREQ starts the dataphone transmitting, and since the transmission is synchronous, nulls (i.e., characters which the other machine will ignore) will be inserted in the data stream, if data characters are not available, to keep the transmission line happy. Thus, until the record is ended, the line is locked in transmission mode and the other machine cannot send a message to the PDP-7. So, in general, transmission should be requested only if it can be foreseen

that the message will be terminated in a reasonable length of time (e.g.,ten seconds), although as long as the message is ended some time no harm will be done.

DFNSND

Purpose: To send the one or two eight-bit characters in the AC to the dataphone, or to end the record.

Calling Sequence: AC = 1 or 2 characters in standard form; or AC bit zero = one; and low-order bits indicate the end-of-record character.

```
        DFNSND
RETURN,———
```

Any eight-bit pattern may be sent over the dataphone in this manner. DFNREQ must have been called as a subtask before DFNSND may be called. For any transmission longer than seventy-three (73) characters (including the end-of-record character) only the first seventy-three characters will be transmitted.

The record will be ended if the high-order bit of the accumulator is set on entry and the low-order eight bits of the accumulator will be used as an end-of-record (EOR) character. If the EOR character is a zero, the default

EOR will be used (ETX). It is recommended that the user choose the default case rather than specify an ETX directly because the bit configuration currently used for ETX may be changed in the near future. After DFNSND is called ending a message, DFNREQ must be called again to start a new message. Another message will be permitted to start when the current message is acknowledged.

DFNSA

Purpose: To send alphanumeric ASCII characters over the dataphone.

Calling Sequence: AC = 1 or 2 characters /IOF or ION

DFNSA

RETURN,——

A carriage return (215) will end a record and will be replaced by an ETX. The first character after the carriage return will start a new record. End-of-page characters (214) and local end-of-files (023) will be sent as independent records, ended by ETX.

DFNRA

Purpose: To read eight-bit alphanumeric character information from the dataphone.

Calling Sequence: DFNRA

RETURN,—— /AC = character

An ETX will be mapped into a carriage return, but all other end-of-record characters will remain the same.

DFNATN

Purpose: To send an "attention" over the dataphone and to clear any blocking conditions.

Calling Sequence: IOF

DFNATN

RETURN,——— /IOF

Table 4. Dataphone Control Characters

CHARACTER	OCTAL	FUNCTION
SOH	201	Start of Header
STX	202	Start of Text
EOT	204	End of Transmission (Hang Up)
ETX	215	End of Text
ATN(ENQ)	005	Attention
ACK	006	Positive Acknowledgment
DLE	220	Data Link Escape
NAK	225	Negative Acknowledgment
SYN	026	Synchronous Idle
ETB	027	End-of-Text Block
EOM	231	End of Message (MTS end-of-file)

2.6 Buffer Management

For those who wish to use the system buffer routines, the following is a brief description of the available routines and of the structure they assume.

Buffers in LOCOSS are linear and fixed in size. The information contained in the buffers may be in either alphanumeric mode or binary mode. In binary mode, all eighteen bits of the word are considered as a whole and are moved about as such. In alphanumeric mode, the lower-order sixteen bits of a word are broken into two eight-bit halves. The right-half byte is always considered significant. The left-half byte is significant only if it is nonzero. When a word is put into a buffer in alphanumeric mode the word is not split up but goes in as a whole. As an option on alphanumeric PUTs, the two bytes will be packed in the buffer if both the last word put into the buffer and the current word being put both consist of only one byte and the last word PUT was nonzero and has not yet been read.

On retrieval, the left-hand byte is considered first and, if significant, is returned to the user. Then the right-hand byte is given to the user. By the calling sequence, the user indicates which mode is to be set in PUTing things into a buffer. The GET routine returns the next thing from the buffer and indicates whether it was PUT in alphanumeric or binary mode.

Each buffer's status is indicated by a five-word Buffer Control Block (BCB). The form of the BCB is as follows:

Word 1: Location of the buffer

Word 2: Flags and Size

Bit 0 = 1 ⇒ Buffer may be packed on alphanumeric PUTs.

Bit 3 = 1 ⇒ Locked to output. Information may not be taken from this buffer.

Bit 4 = 1 ⇒ Locked to input. Information may not be put into this buffer.

Bit 5 = 1 ⇒ Buffer in binary mode.

Bit 6 = 1 ⇒ Buffer in alphanumeric mode.

Bit 5 = Bit 6 = 0 ⇒ Buffer empty.

Bit 6 - 17 = Size of buffer (max = 3777 octal).

Word 3: PI—Inbound Pointer. The last word put into the buffer (if any) went into this location. Thus the next word goes into location PI + 1.

Word 4: PO—Output Pointer. The next word to be taken from the buffer (if non-empty) is at address PO + 1.

Word 5: Temporary Storage—for the right-hand character in an alphanumeric fetch.

For example, the initial BCB for the teleprinter routines looks like:

```
TPRBCB, TPRBUF      /location of a buffer
        400017      /of size 15, originally empty, lock
                    /bits off,packing allowed
        TPRBUF      /PI
        TPRBUF      /PO
        Ø           /temporary storage
```

The following routines clear a buffer, get from and put into it, and lock and unlock it to input and output. The user need only reserve the buffer area and set up the original BCB.

The routines then do all the work. Note that all buffer control routines must be called with interrupts off, and that return is made with interrupts off.

CLEARB

Purpose: To reset a buffer control block to indicate that the buffer is empty.

Calling Sequence:
```
        IOF
        CLEARB
        BCB
RETURN, ------
```

BCB is the address of the associated buffer control block.

PUT

Purpose: To put a word into a buffer.

Calling Sequence:
```
        LINK=MODE
        AC=WORD
        IOF
        PUT
        BCB
ERRET,  AC=WORD
        ------
RETURN, ------
```

Where: Mode = 1 for binary put
= 0 for alpha put

BCB is the address of the associated buffer control block. If the buffer is full, locked in, or already contains information

in a different mode than in the calling sequence, control is returned to location ERRET with the AC=WORD and the word not having been put into the buffer. If the word has been put into the buffer, control is transferred to location RETURN. Note that the error condition has a two-word return, i.e., RETURN= ERRET+2.

GET

Purpose: To retrieve the next item from a given buffer.

Calling Sequence:

```
        IOF
        GET
        BCB
ERRET,  ---
        ---
RETURN, ---     /AC = ITEM, LINK=MODE
```

Where: BCB is the associated buffer control block

ERRET is the location transferred to if the buffer is empty or locked out.

RETURN is the location transferred to on successful retrieval.

MODE = 1 ➢ the item returned is binary (18 bits)

= 0 ➢ the item returned is alphanumeric (8 bits)

ITEM is the available information from the buffer.

NOTE: The low-order 16 bits of an alphanumeric word taken from the buffer are split into two eight-bit halves or bytes and given one at a time, left byte first, to the user.

However, if the left byte is all zeros it is ignored and the right is given immediately. The right byte is always returned whether it is zero or not. This convention is to allow characters to be put into the buffer either one or two at a time.

LOCKBI

ALOWBI

LOCKBO

ALOWBO

Purpose: To allow and disallow transfers to and from buffers.

Calling Sequence:

```
IOF
LAW BCB + 1     /address of BCB status word
XXXXXX
```

Where: XXXXXX is one of the routines:

LOCKBI - Lock buffer to input, i.e., disallow PUTs.

ALOWBI - Allow buffer input.

LOCKBO - Lock buffer to output, i.e., disallow GETs.

ALOWBO - Allow buffer output.

APPENDIX A

LOCOSS - USER SYMBOLS

BLANK PAGE

Appendix A. LOCOSS User Symbols

```
LOCOSS-3 USER SYMBOLS--28AUG68


SUBTSK=JMS I 24          /INSERT A PROGRAM AS A SUBTASK
INSERT=JMS I 25          /INSERT AN INDEPENDENT TASK
BUSY  =JMS I 26          /COME BACK LATER
DELETE=JMP I 27          /DELETE THE CURRENT TASK
CREATE=JMS I 30          /CREATE A UNIQUE EVENT NUMBER
WAIT  =JMS I 31          /WAIT FOR AN EVENT TO OCCUR
SIGNAL=JMS I 32          /SIGNAL THE OCCURRENCE OF AN EVENT
INTABL=JMS I 33          /ENTER INTERRUPT HANDLER INTO INTERRUPT TABLE
RDRA  =JMS I 34          /GET ALPHANUMERIC CHARACTER FROM READER
RDRB  =JMS I 35          /GET BINARY WORD FROM READER
RDRI  =JMS I 36          /INITIALIZE TAPE READER
PCHA  =JMS I 37          /PUNCH ALPHANUMERIC CHARACTER
PCHB  =JMS I 40          /PUNCH BINARY WORD
PCHI  =JMS I 41          /INITIALIZE TAPE PUNCH
TPRA  =JMS I 42          /TYPE ALPHANUMERIC CHARACTERS
TPRI  =JMS I 43          /INITIALIZE TELEPRINTER ROUTINES
KBDA  =JMS I 44          /READ ALPHANUMERIC CHARACTER FROM KEYBOARD
KBDI  =JMS I 45          /INITIALIZE KEYBOARD ROUTINES

CLEARQ=JMS I 47          /CLEAR THE TASK QUEUE
CLEARE=JMS I 50          /REINITIALIZE EVENT HANDLING
CLEARB=JMS I 51          /CLEAR A BUFFER CONTROL BLOCK
RESET =JMS I 52          /REINITIALIZE ALL OF LOCOSS
GET   =JMS I 53          /GET A CHARACTER FROM A BUFFER
PUT   =JMS I 54          /PUT A CHARACTER INTO A BUFFER
RDRSEL=JMS I 55          /START THE TAPE READER
PRMPTC=      56          /PROMPTING CONTROL; 0 => GIVE PROMPT
ECHOSW=      57          /ECHO CONTROL; 0 => ECHO KEYBOARD
TABCNT=      60          /TAB STOP SETTING
KBDLOC=      61          /LOCATION OF KEYBOARD BUFFER CONTROL BLOCK
DFNRED=JMS I 62          /GET CHARACTER FROM DATAPHONE
DFNSND=JMS I 63          /SEND A CHARACTER TO THE DATAPHONE
DFNREQ=      64          /SEIZE DATAPHONE FOR TRANSMISSION

DISK  =JMS I 66          /TALK TO IBM 1800 INTERRUPT ROUTINE
DSKERR=JMS I 67          /SET ERROR RETURN FROM "DISK" CALL
DSKGET=JMS I 70          /GET A CHARACTER FROM A DISK FILE
DSKPUT=JMS I 71          /PUT A CHARACTER INTO A DISK FILE
LOCKBO=JMS I 72          /LOCK BUFFER TO OUTPUT
ALOWBO=JMS I 73          /ALLOW BUFFER OUTPUT
LOCKBI=JMS I 74          /LOCK BUFFER TO INPUT
ALOWBI=JMS I 75          /ALLOW BUFFER INPUT
CLOCK =JMS I 76          /SET RETURN FROM INTERVAL TIMER
LOAD  =JMS I 77          /LOAD A FILE FROM THE DISK
```

```
DFNI   =JMS I 100          /INITIALIZE DATAPHONE
CLOCKI=JMS I 101           /INITIALIZE CLOCK ROUTINES
RDRLOC=      102           /LOCATION OF READER BUFFER CONTROL BLOCK
PCHLOC=      103           /LOCATION OF PUNCH BUFFER CONTROL BLOCK
DFNSA =JMS I 104           /SEND ALPHANUMERIC CHARACTER TO DATAPHONE
DFNRA =JMS I 105           /READ ALPHANUMERIC CHARACTER FROM DATAPHONE
DUMMY =JMS I 106           /DUMMY ROUTINE. BOTH SOURCE AND SINK, TOO.
RDOCT =JMS I 107           /READ OCTAL NUMBER
RDDEC =JMS I 110           /READ DECIMAL NUMBER
PROCT =JMS I 111           /OUTPUT OCTAL NUMBER
PRDEC =JMS I 112           /OUTPUT DECIMAL NUMBER
DFNATN=      113           /SEND ATTENTION OVER DATAPHONE
SYSDSK=JMS I 114           /TALK TO IBM 1800, WITH IMMEDIATE ERROR RETURN
ASCOUT=JMS I 115           /OUTPUT A LIST
ICYRDT=JMS I 116           /READ FROM IBM 1800 COPY PORT TRANSLATED
ICYWRT=JMS I 117           /WRITE TO IBM 1800 COPY PORT TRANSLATED
ICYINI=JMS I 120           /INITIALIZE IBM COPY PORT ROUTINES
TRMCHR=      121           /TERMINATING CHARACTER FOR RDOCT AND RDDEC
SOURC1=      122           /ASSIGNABLE SOURCES
SOURC2=      123
SOURC3=      124
SINK1 =      125           /ASSIGNABLE SINKS
SINK2 =      126
SINK3 =      127
LEDING=      130           /LEADING ZERO REPLACEMENT
SEZTPR=      131           /SEIZE TELEPRINTER ROUTINES

ICYRDU=JMS I 133           /READ FROM IBM 1800 COPY PORT UNTRANSLATED
ICYWRU=JMS I 134           /WRITE TO IBM 1800 COPY PORT UNTRANSLATED

/DISK OPERATOR DEFINITIONS

D.ECHO=00400               /01 ECHO
D.OPOT=01000               /02 OPEN OUTPUT FILE TRANSLATED
D.OPIT=01400               /03 OPEN INPUT FILE TRANSLATED
D.PUT =02000               /04 PUT TO OUTPUT FILE
D.GET =02400               /05 GET FROM INPUT FILE
D.CLO =03000               /06 CLOSE OUTPUT FILE
D.CLI =03400               /07 CLOSE INPUT FILE
D.CLR =04000               /08 CLOSE ALL FILES AND CLEAR
D.CRE =04400               /09 CREATE FILE
D.DES =05400               /11 DESTROY FILE
D.OPOU=06400               /13 OPEN OUTPUT FILE UNTRANSLATED
D.OPIU=07000               /14 OPEN INPUT FILE UNTRANSLATED
D.ECOA=10000               /16 EBCDIC TO ASCII ECHO
D.ECOB=10400               /17 ASCII TO EBCDIC ECHO
D.BOOT=11000               /18 BOOTSTRAP STYLE LOAD
D.REDT=11400               /19 READ FROM COPY PORT TRANSLATED
D.WRIT=12000               /20 WRITE TO COPY PORT TRANSLATED
D.REDU=12400               /21 READ FROM COPY PORT UNTRANSLATED
D.WRIU=13000               /22 WRITE TO COPY PORT UNTRANSLATED

FIX
PAUSE
```

LOCOSS-3 USER SYMBOLS--28AUG68

ALOWBI	120075	ALOWBO	120073	ASCOUT	120115	BUSY	120026
CLEARB	120051	CLEARE	120050	CLEARQ	120047	CLOCK	120076
CLOCKI	120101	CREATE	120030	DELETE	620027	DFNATN	113
DFNI	120100	DFNRA	120105	DFNRED	120062	DFNREQ	64
DFNSA	120104	DFNSND	120063	DISK	120066	DSKERR	120067
DSKGET	120070	DSKPUT	120071	DUMMY	120106	D.BOOT	11000
D.CLI	3400	D.CLO	3000	D.CLR	4000	D.CRE	4400
D.DES	5400	D.ECHO	400	D.ECOA	10000	D.ECOB	10400
D.GET	2400	D.OPIT	1400	D.OPIU	7000	D.OPOT	1000
D.OPOU	6400	D.PUT	2000	D.REDT	11400	D.REDU	12400
D.WRIT	12000	D.WRIU	13000	ECHOSW	57	GET	120053
ICYINI	120120	ICYRDT	120116	ICYRDU	120133	ICYWRT	120117
ICYWRU	120134	INSERT	120025	INTABL	120033	KBDA	120044
KBDI	120045	KBDLOC	61	LEDING	130	LOAD	120077
LOCKBI	120074	LOCKBO	120072	PCHA	120037	PCHB	120040
PCHI	120041	PCHLOC	103	PRDEC	120112	PRMPTC	56
PROCT	120111	PUT	120054	RDDEC	120110	RDOCT	120107
RDRA	120034	RDRB	120035	RDRI	120036	RDRLOC	102
RDRSEL	120055	RESET	120052	SEZTPR	131	SIGNAL	120032
SINK1	125	SINK2	126	SINK3	127	SOURC1	122
SOURC2	123	SOURC3	124	SUBTSK	120024	SYSDSK	120114
TABCNT	60	TPRA	120042	TPRI	120043	TRMCHR	121
WAIT	120031						

APPENDIX B

SYSTEMS INFORMATION

APPENDIX B

SYSTEMS INFORMATION

The information in this section is meant to provide assistance in understanding the program listings and thus to aid program maintenance. In general, LOCOSS is a collection of subroutines and, as such, the description of the individual subroutines and the general comments at the head of each section provide most of the information needed to specify completely the action taken by an individual routine. In these cases the comments in the listings and the description in the user's manual will not be supplemented here. There are, however, four sections of interest in which there is an underlying structure not completely specified in the user's manual and which must be understood in order to make sense from the listings. These are: the structure of the task queue, the structure of the event table, the mechanism of the Command Language Interpreter, and the conventions for information transfer across the IBM 1800-PDP-7 interface. An additional topic, the structure of the loaders, is discussed in the section on core image formats.

TASK QUEUE

The task queue is a fixed-length buffer containing a linked list of task control blocks (TCBs). A TCB contains the information needed to start some program as a task. It consists of five words with the following format.

Word 1: Link pointer to the next task in the queue.

Word 2: Event control word.

Bit Ø: =Ø This TCB is not part of an event chain, i.e., it was put into the queue by INSERT, SUBTSK, or BUSY, and may be started in the normal manner.

=1 This TCB is part of an event chain. Do not activate it until bit zero is zero.

Word 3: Activation word.

Bit Ø: =Ø Enter as a JMS when the task is enterable.

=1 Enter as a JMP when its turn comes (always enterable).

Bits 5-17: Location of program entry point.

Word 4: Return pointer, Bits 0-17, meaningful only for JMS entry.

Word 5: Parameter word, Bits 0-17. This was the contents of the accumulator when the multiprogramming system was called and will be restored before control is returned to the user program.

Calls to INSERT and SUBTSK cause the activation word to be set wholly from the location following the calling instruction so that the user has control over the method of entry to the addressed task (by setting the high-order bit as desired).

BUSY sets the high-order bit of the activation word so that return will be made via a JMP entry. These three calls all zero the event control word. A call on WAIT sets the activation word as in BUSY, sets bit zero of the event control word to one, and sets the rest of the word as described below in the event descriptions.

The multiprogramming system maintains three variables which describe the state of the queue. QHEAD contains the address of the first TCB in the linked list, i.e., the next TCB to be activated. QTAIL contains the address of the last TCB on the list, i.e., the point at which another TCB may be added to the list. QFREE contains the address of the first unused TCB in a linked list of free storage.

Upon initialization, all of the area used for the queue is split into five-word blocks, and linked together, forming the free storage list. The first word of each block is a pointer to the next block, and the end of the list is marked by a zero pointer. One five-word block is reserved as the "idle task." This TCB has its event control word set so that it is never activated. Its link pointer points to itself so that the list forms a ring. Both QHEAD and QTAIL point to the idle task TCB.

The task control loop operates as follows. The TCB addressed by QHEAD is considered for activation. If the TCB may not be activated it is put at the end of the queue; that is, the pointers QHEAD and QTAIL are changed so that QTAIL points to the rejected task and QHEAD points to the next task

in the queue (the TCB addressed by the link pointer of the rejected TCB). If the TCB may be activated (i.e., the event control word is not set and the task is enterable), it is removed from the queue, and control is transferred to the task in the appropriate manner.

Removal from the queue consists of closing up the link pointers and adding the used TCB to the free storage list. Used TCBs are linked to the head of the free storage list and new TCBs are obtained also from the head of the free storage list. A new TCB is created by calls to INSERT, SUBTSK, BUSY, and WAIT, and is added to the queue by obtaining a TCB from the free storage list and linking it into the queue after the TCB addressed by QTAIL. The link pointer of the new TCB is set to point to the TCB addressed by QHEAD. Thus the queue is always maintained as a ring of TCBs. At initialization time the ring consists of one TCB which can never be removed from the list, and which points to itself. Insertions and deletions from the queue, then, only expand and contract the size of the ring.

If at any point a new TCB is needed and the free storage head, QFREE, points to a zero word, there is no room in the inn and control is passed to the LOCOSS error handler.

EVENT CONTROL

The event table consists of a fixed number of Event Control Blocks (ECBs), consisting of two words each. If the first word of the ECB is zero the block is not being used. If the

first word is nonzero then its contents (a number) name an event which has been created and which has not yet been signaled. The second word of the block, then, is a pointer to the head of a linked list of Task Control Blocks (TCBs) naming tasks which have called subroutine WAIT with the event named as parameter; i.e., these are tasks awaiting the occurrence of the named event. If the second word of the ECB is zero, then no tasks have yet executed calls on WAIT.

When a user program calls CREATE, a unique number for an event is produced by using the contents of location EVNO. After this number is used, the contents of EVNO are incremented by one so that the next call on CREATE will produce a different number. A search is then made of the event table, looking for an ECB with a first word zero (i.e., an unused ECB). If none is found, control is passed to the error processor. If an empty ECB is found, the new event number is stored in its first word establishing it as an occupied ECB; the second word is zeroed, indicating that no tasks are waiting for the event.

When a task calls WAIT, a search is made of the event table for the event given in the call. If an ECB for the named event does not exist, it is assumed that the event has already occurred or that it has not been created yet, and control is returned immediately to the user. If the event has an ECB, a TCB is created for the task and added to the task queue. The event control word of the TCB (word two) is then established as the head of the linked list of TCBs waiting

for that event to take place. The end of the list is indicated by Bits 1-17 of the event link-pointer (TCB word two) being zero. Word two of the ECB is then set to point to the head of the list of TCBs. Note that the list is a sequence of TCB-word-twos, not of the TCBs themselves.

When a program calls SIGNAL, a search is made of the event table. If the event named has no ECB, return is immediate to the calling program, since the event has already taken place, or was never really created. If the event does have an ECB, the list of TCBs is threaded, zeroing all the event control words, thus enabling those TCBs to be reactivated the next time they come up for reconsideration. The first word of the ECB is then zeroed, indicating that the event has occurred. Any further WAITs or SIGNALs will have immediate returns since no ECB exists for that event.

THE COMMAND LANGUAGE INTERPRETER

The CLI is a straightforward command processor which cues on three-letter commands and dispatches to the specific handler for each command. For those commands taking numeric parameters, the CLI uses "RDOCT" and "RDDEC" routines very similar to the ones available to users and described in the section on general routines. For the device assignment commands, the syntax is enforced by restricting the portion of the command table which will be searched.

The CLI is an interrupt-time package. This means that any command entered via the keyboard is acted upon as soon as the keyboard routines receive the interrupt from the entry of

the carriage return ending the line. This feature ensures that the CLI is always available to the user and that the user may be sure that his request is acted upon as soon as it is recognized; that is, the CLI does not have to compete with task time for CPU control or for system resources (subroutines, teleprinter, etc.). However, there are some command functions which require that interrupts be turned on. For example, a DMP command may require the output of several lines of numbers. Interrupts may not be turned off during the time-consuming output of these numbers since there is at least one synchronous device (the dataphone) which must run under interrupt control. To achieve this capability the CLI essentially "pushes down" task-time control and establishes itself as the active task. More specifically the CLI saves all the interrupt-saved registers and then turns interrupts on. If an interrupt occurs during CLI processing, the interrupt identifier saves the registers as usual and then processes the interrupt normally. Upon completion of the servicing, the interrupt processor restores the registers for the CLI and returns control to the CLI. When the CLI is done processing, it restores the registers it has pushed down and transfers control to the interrupted task-time program.

The only possible source of embarrassment which may occur is re-entry to the CLI while it is already executing. This nuisance is remedied by cutting it off at the source—the keyboard dispatcher. If a keyboard interrupt occurs while the CLI is executing, the registers which were pushed down are

immediately popped up so that the interrupt processor will use these first-level registers upon return from the keyboard processing, and thus return control to the task-time routine originally interrupted. Thus, a keyboard interrupt will abort CLI activity; and, since it is only by keyboard activity that the CLI may be invoked, the CLI will never be invoked while already running.

IBM 1800 COMMUNICATION ROUTINES

All communication with the IBM 1800 across the "minor" interface is performed on a request basis (with one exception*) with the PDP-7 initiating all information transfers. The data path from the PDP-7 to the 1800 is sixteen bits wide. Of this, the high-order eight bits are used to specify a function to be performed by the 1800 service routine. The low-order eight bits are treated as data according to the specific function requested. In general, the data may be an eight-bit character, a file number, or immaterial.

Upon receipt of the information from the 7, the 1800 attempts the function requested and, in the twelve-bit path from the 1800 to the 7, returns either a four-bit positive or negative acknowledgment to the request and an eight-bit data byte. Again, depending on the function, the datum may be a character,

---

* Brender, R.F., Frantz, D.R., Foy, J.L. Jr., and Schunior, T.W., Specialized System Software for Interacting DEC PDP-7 and IBM 1800, Technical Report 11, Concomp Project, University of Michigan, Ann Arbor, October 1968.

an error code (in case of negative acknowledgment), or it may be meaningless.

To check for possible failure of the interface in actually transferring information, the disk routines perform time-outs on the two directions of information transfer. Thus, if the 1800 is not listening (i.e., does not have a pending read), the disk routines will wait about a second and a half, and then return an error code to the user's program.

BIBLIOGRAPHY

PDP-7 User's Handbook, DEC No. F-75, Digital Equipment Corporation, Maynard, Mass., 1965.

Mills, D., RAMP: A PDP-8 Multiprogramming System for Real-Time Device Control, Technical Memorandum, Concomp Project, University of Michigan, Ann Arbor, May 1967, 24 pp.

Wood, D.E., A 201A Data Communication Adaptor for the PDP-8, Memorandum 15, Concomp Project, University of Michigan, Ann Arbor, February 1968, 134 pp.

Brender, R.F., Frantz, D.R., Foy, J.L. Jr., and Schunior, T.W., Specialized System Software for Interacting DEC PDP-7 and IBM 1800, Technical Report 11, Concomp Project, University of Michigan, Ann Arbor, October 1968.

Brender, R.F., and Foy, J.L. Jr., Flexible High-Speed Interface between IBM 1800 and DEC PDP-7 Computers, Technical Report 12, Concomp Project, University of Michigan, Ann Arbor, October 1968.

MTS: Michigan Terminal System, 2nd ed., Computing Center, and Concomp Project, University of Michigan, Ann Arbor, December 1967, 2 vols.

Mills, D.L., The Data Concentrator, Technical Report 8, Concomp Project, University of Michigan, Ann Arbor, May 1968, 113 pp.

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
THE UNIVERSITY OF MICHIGAN
CONCOMP PROJECT

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b GROUP

3 REPORT TITLE
LOCOSS: A Multiprogramming Monitor for the DEC PDP-7.

4 DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Technical Report 10

5 AUTHOR(S) *(Last name, first name, initial)*
D.R. Frantz, R.F. Brender, and J.L. Foy, Jr.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
November 1968	120	

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
DA-49-083 OSA-3050 b. PROJECT NO.	Technical Report 10
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. AVAILABILITY/LIMITATION NOTICES
Qualified requesters may obtain copies of this report from DDC.

11 SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Advanced Research Projects Agency

13 ABSTRACT

LOCOSS (Logic of Computers Operating System for the PDP-Seven) was developed to provide a suitable run-time environment in which to run applications programs. Multiprogramming capabilities are an essential part of the system and allow a flexible organization. Provision is made for alternate methods of establishing a separate (parallel) line of execution and for invoking a subtask. EVENTS provide a flexible means of intertask communication. A keyboard Command Interpreter provides a number of real-time control services and simple debugging aids. Input/output is buffered, overlapped, and essentially device-independent, allowing users to write programs with references to generalized "sources" and "sinks." I/O devices supported are: teletype, paper tape reader and punch, 201A dataphone, IBM 1800 (the disk file system and running programs), and the 337 display as a character sink. Additional subroutines are available for such user needs as reading and writing octal or decimal numbers, loading a program from the IBM 1800 disk, and setting an interval timer.

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
PDP-7						
Multiprogramming						
Monitor						
Executive System						
Small Computers						
Keyboard Monitor						

INSTRUCTIONS

I. ORIGINATING ACTIVITY: Enter the nsme snd addresa of the contractor, subcontrsctor, grantee, Depsrtment of Defenae activity or other organization *(corporate suthor)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall aecurity clssaificstion of the report. Indicste whether "Restricted Data" is included. Marking Is to be in sccordance with appropriate aecurity regulstions.

2b. GROUP: Automatic downgrading is apecified in DoD Directive 5200. I0 snd Armed Forces Industrisl Manual. Enter the group number. Also, when spplicsble, ahow that optional markinga have been uaed for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capitsl lettera. Titles In all csses should be unclaasified. If s mesningful title csnnot be selected without clasaificstion, ahow title clasaification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriste, enter the type of report, e.g., interim, progreaa, summary, annual, or final. Give the inclusive datea when a apecific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as ahown on or in the report. Enter lsst nsme, firat nsme, middle initial. If military, show rank snd branch of aervice. The nsme of the principal suthor is sn abaolute mininum requirement.

6. REPORT DATE: Enter the dste of the report aa day, month, yesr, or month, yesr. If more than one date appesrs on the report, use date of publicstion.

7a. TOTAL NUMBER OF PAGES: The total page count ahould follow normal pagination procedurea, i.e., enter the number of pagea containing informstion.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If sppropriate, enter the applicable number of the contract or grant under which the report wss written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the sppropriate militsry department identificstion, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the offlcisl report number by which the document will be identified and controlled by the originating sctivity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report hss been assigned sny other report numbers *(either by the originator* or *by the sponsor)*, slso enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter sny limitations on further diasemination of the report, other than those imposed by aecurity clasaification, using standard ststements such ss:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and disaemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of thIs report directly from DDC. Other qualified DDC usera shall request through

_____."

(4) "U. S. military sgencies may obtain copies of this report directly from DDC. Other qualified users shsll request through

_____."

(5) "All distribution of this report is controlled. Qualified DDC usera shall request through

_____."

If the report has been furnished tc the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

II. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmentsl project office or laboratory sponsoring *(paying for)* the reasearch and development. Include sddress.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may slso appear elsewhere in the body of the technicsl report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of clsssified reports be unclasaified. Each paragraph of the abstract shall end with an indication of the military security classification of the informstion in the paragraph, represented as *(TS), (S), (C), or (U)*.

There is no limitation cn the length of the sbstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words sre technically mesningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security clsssificstion is required. Identifiers, such ss equipment model desisnation, trade name, militsry project code name, geographic locstion, may be used as key words but will be followed by sn indication of technicsl context. The sssignment of links, rules, and weights is optionsl.